青少年计算机竞赛指导丛书

青少年国际和全国信息学（计算机）奥林匹克竞赛指导

——图论的算法与程序设计

吴文虎 王建德 编著



清华大学出版社

青少年计算机竞赛指导丛书

青少年国际和全国信息学（计算机）奥林匹克竞赛指导

——图论的算法与程序设计

吴文虎　王建德　编著

清华大学出版社

(京)新登字 158 号

内 容 简 介

本书介绍了有关图论的基本概念和典型的图论算法，结合历届赛题分析如何把一个实际问题抽象化为数学中的图论问题，并给出了程序解法。本书的特点是既有基本概念的讲解及对所解问题的分析，又有编程的思路与参考程序。是参加国际和全国奥林匹克竞赛学生的必读书，也可作为大学生的参考书。

图书在版编目(CIP)数据

青少年国际和全国信息学(计算机)奥林匹克竞赛指导：图论的算法与程序设计/吴文虎，王建德编著. —北京：清华大学出版社，1996.8
(青少年计算机竞赛指导丛导)
ISBN 7-302-02237-2

Ⅰ. 青… Ⅱ. ① 吴… ② 王… Ⅲ. 图论算法-程序设计-中学-计算机课-教学参考资料 Ⅳ. G634.673

中国版本图书馆 CIP 数据核字(96)第 13326 号

出版者：清华大学出版社 (北京清华大学校内，邮编 100084)
印刷者：人民文学印刷厂
发行者：新华书店总店北京科技发行所
开　本：787×1092 1/16　印张：11　字数：258 千字
版　次：1997 年 3 月第 1 版　1997 年 3 月第 1 次印刷
书　号：ISBN 7-302-02237-2/TP・1087
印　数：0001—5000
定　价：12.00 元

前　言

信息科学与技术正在对人类社会的发展产生难以估量的深远影响，并将成为新世纪的一个标志。作为人类总体智慧的结晶，电脑已经成为一种新的现代文化。“计算机的普及要从娃娃做起”已经成为“科教兴国”的一项重要内容。

一个国家、一个民族要想不落伍，要想跻身于世界先进民族之林，关键在于拥有高素质的人才；综合国力的竞争，说到底也是人才的竞争。电子计算机是现代科学与技术的基础和核心，它的飞速发展，把社会生产力水平提到前所未有的高度，人类进入了信息时代。电脑对人类社会的发展所起的巨大作用，特别是对人类智能的发展所起的促进作用，已为人们普遍认识到。计算，跟 语言一样是人类社会每时每刻都不可缺少的。现在，人类已经拥有了帮助自己进行复杂计算与思维的工具，电子计算机起到了人脑延伸的作用。以往历史上的技术革命，只能起到创造和改进工具，用机器代替人的体力的作用；而计算机则是把人从重复性的或有固定程式的脑力劳动中解放出来，使自己的智能获得空前的发展。作为“人类通用智力工具”计算机在开发人类智能方面所起的无与伦比的作用不容忽视。这也就是计算机与基础教育相结合，能够成为当今世界的大趋势的一个原因。从信息社会要求人才具备的科学素养看，数学、物理学、化学、生命科学和信息科学是五大支柱，这正是联合国教科文组织倡导举行五项国际学科奥林匹克竞赛的内容。

国际信息学奥林匹克(International Olympiad in Informatics，简称 IOI)始于 1989 年，到 1996 年已成功地举办了 8 届。这是一种智力与应用计算机能力的大赛，从益智的角度看，是用电脑帮助开发人脑，重在提高思维能力，培养创新意识。在中国队的训练中强调德智体美全面发展；心态上自立、自尊、自信、自强，要怀着中华民族的自豪感和自信心去参赛，这种心态是学习、训练和取胜的重要条件。8 届比赛，中国队每届都取得了名列前茅的好成绩，31 人次参赛，夺得 31 块奖牌，其中金牌 17 块、银牌 6 块、铜牌 8 块。特别是 IOI'95(荷兰)突破了前 6 届比赛女孩与金奖无缘的纪录，两名中国女选手荣登金牌领奖台。在 IOI'96(匈牙利)上中国队又实现了全金的突破，四名选手，每人夺得 1 块金牌。

从大局看，竞赛不是目的，是推动普及的手段，我们的目的只有一个：“科教兴国”。竞赛活动带有因材施教、因材施测的特点。普及是有层次的，与学科竞赛有关的普及活动，对青少年而言属于比较高的层次，当然就有相当的难度。我们编写的这套青少年计算机竞赛指导丛书，涉及程序设计语言、常用算法、组合数学、图论、人工智能搜索等的基本知识和基本方法。这些理论知识往往都是通过竞赛当中的一些实例来讲解的。目前已拟就了三本书：其一是《组合数学的算法与程序设计》，其二是《图论算法与程序设计》；其三是《国际国内青少年信息学(计算机)竞赛试题解析(1994～1995)》，前两本书试图从数学的基本概念和基本理论出发，给出算法设计和编写程序的方法，第三本书收集了最近国际国内比赛的典型试题及其解法，重点放在解题思路上。这里许多题目比较新颖，很难去套固定算法或固定模式，这中间有些招数是选手们想出来的。从中可以看出信息学奥林匹克要求创

新，鼓励创新。当然，书中给出的解法，对青少年读者而言，我们希望仅仅起到抛砖引玉的作用，并且热切盼望引出更多的玉来。作为老师，我和王建德都这样想，"精心育桃李，热望青胜兰"就是我们编写这套丛书的初衷。

中国计算机学会普及委员会主任
国际信息学奥林匹克中国队总教练
清华大学计算机科学与技术系教授
吴文虎
1996 年 11 月 20 日

目　录

第一章 基本概念

1.1 引　言

图论是一个应用十分广泛而又极其有趣的数学分支。物理、化学、生物、科学管理、计算机等各个领域都可找到图论的足迹。本书不想对图论作广泛而深入的探讨，主要介绍一下图论的一些基本知识、图论中常用的初等方法和典型的图论程序，试图在抽象理论和具体编程之间为读者架设一座桥梁。

为了让大家知道这本书主要讲些什么，我们先举几个例子。

【例 1】 图 1-1 画的是一个"图"，当然究竟什么叫做"图"，以后还要仔细讲。这里只要先记住，我们研究的图，指的是由顶点和线组成的图形。我们先把图 1-1 看成是一个公路网，$V_1, V_2, \cdots, V_{10}$是一些城镇，每条线旁边的数字代表这一段公路的长度。现在问，要从 V_1 把货物运到 V_{10}，走哪条路最近？

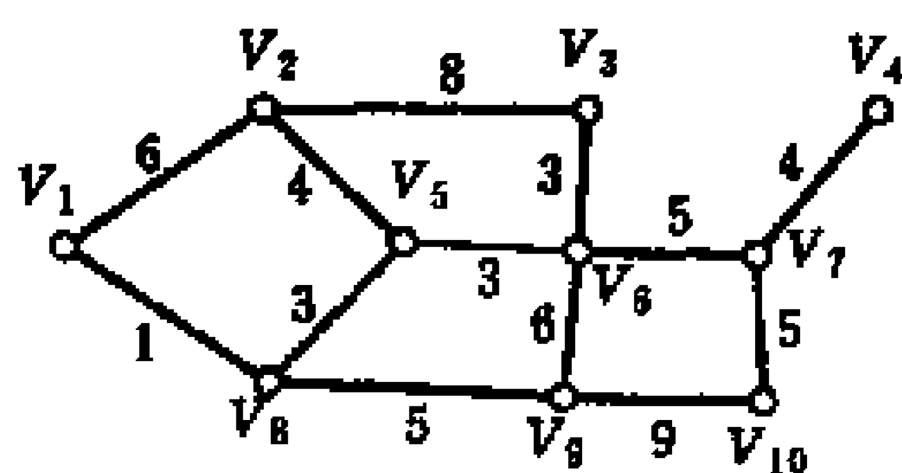


图　1-1

这个问题通常叫做最短路径问题。读者不难看出，这是一个有很大现实意义的问题，它不仅出现在各种运输问题中，而且在电路设计等问题中也有用。因为这种问题研究的是从 V_1 到 V_{10}的所有路径中，哪一条路最短？因此它是一个极大极小问题，即极值问题，而它又和一个"图"密切联系着，因此这种问题就叫做图论中的极值问题。

【例 2】 还是把图 1-1 看成公路网，$V_1, V_2, \cdots, V_{10}$看成公路网的一个站点，若这个公路网目前被敌方占领。请分析一下，能否仅破坏其公路网的一个站点或者至少破坏敌人哪几个站点，就可摧毁敌方整个运输线。

这类问题称为图的连通性问题。原来的公路网，任意两个站点之间互相可达，这样的图称作连通图。一旦删去一些(或一个)点以及和它们关联的边时，这张图就不再成为一张完整的连通图了。因此也有人把这种问题叫做"割点"问题。军事指挥中这类问题就很多。

上面二个问题都是明显地和一个图联系着的。让我们再来讲一个例子，从表面上看，它与图并没有什么关系，但经过仔细分析，可以把它归结为图论中的一个匹配问题。

【例 3】 飞行大队有若干个来自各地的驾驶员，专门驾驶一种型号的飞机，这种飞机每架有两个驾驶员。由于种种原因，例如相互配合的问题，有些驾驶员不能在同一架飞机上飞行，问如何搭配驾驶员，才能使出航的飞机最多。

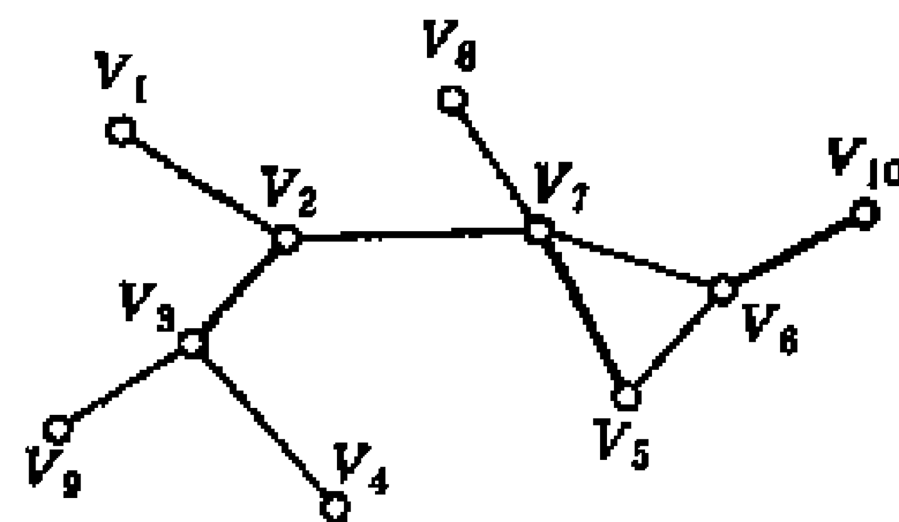


图　1-2

为简单起见，假设有 10 个驾驶员，图 1-2 中的 V_1，

V_2、…，V_{10}就代表这 10 个驾驶员。如果两个人可以同机飞行，就在代表他们两个之间连一条线；两个人不能同机飞行，就不连。例如 V_1 和 V_2 可以同机飞行，而 V_1 和 V_3 就不行。画了这个图后，就可以研究搭配飞行员的问题了。图 1-2 中画的 3 条粗线就代表了一种搭配方案。由于一个飞行员不能同时派往两架飞机，因此任何两条粗线不能有公共的端点，今后我们把一个图中没有公共端点的一组线叫做一个"匹配"。这样定义以后，上面问题就成为：如何找一个包含最多线的匹配？这个问题叫做图的最大匹配问题。请大家试试看，能不能从图 1-2 中找出一个包含 4 条线的匹配，再试试能不能找到包含 5 条线的匹配。

像上述例子那样，将实际生活中的事物分析转化为图论问题的实例还很多，后面还要讲，这里就不多介绍了。

现在可以把这本书的目的说一说了。

这本书主要是围绕一些有趣的数学难题、计算机竞赛试题展开对图论的讨论，讲它们是怎么从实际生活中提炼出来的，怎样通过编程求解这些问题。哪些问题已有精确的解答，哪些仅是近似算法。当然本书不可能穷尽图论知识，只能选一些比较基本的来讲讲。重点是放在编程解题和实际应用上。限于篇幅，对一些编程过程中未涉及的抽象定理只能做简单的介绍而不再推导。市面上有关图论知识的教科书和普及读物甚多，有兴趣的读者可查阅这方面的资料。

1.2 图的定义

图论研究的对象是图，什么是图呢？

图 1-3 就是一个图，它有若干个不同的点 V_1，V_2，…，V_{11}，我们称之为顶点。这些顶点中有一些是用直线段或曲线段连接的，我们把这些直线段和曲线段称作边。例如 V_1 与 V_2 之间有两条边。若连接两个顶点的边有多条，则这些边称之为平行边。V_2 与 V_3 之间有一条边，V_2 与 V_4 之间没有边……等等。图 1-3 中，V_1 与 V_1 本身也有边相连，这样的边叫做环。当然，也可能出现某顶点与图中除它外的每一顶点均不相连的情况，这种顶点称为孤立点，例如 V_{11}。

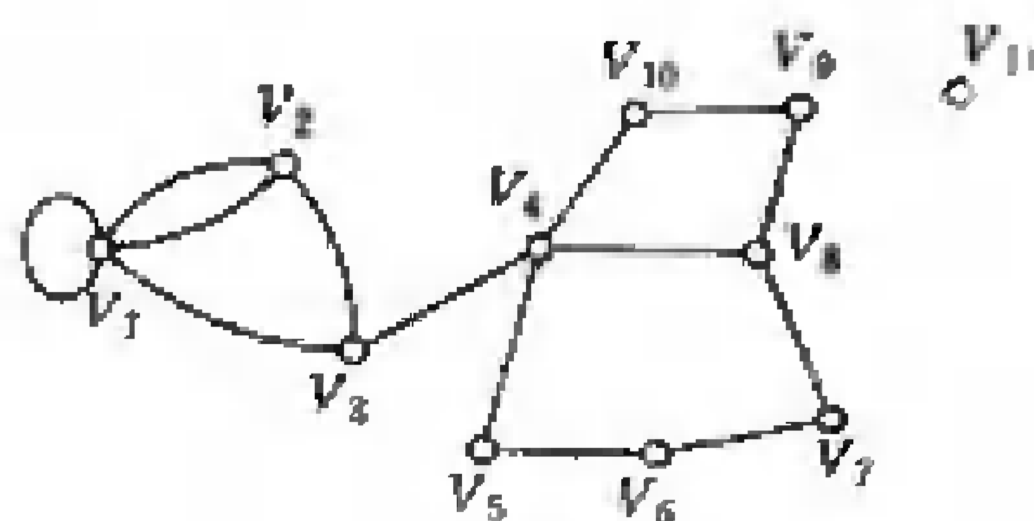


图 1-3

由此得出图的定义：

由若干个不同顶点与连接其中某些顶点的边所组成的图形就称为图。

要注意的是，在图的定义中，顶点的位置以及边的曲直长短都是无关紧要的，而且也没有假定这些顶点和边都要在一个平面内(譬如说，正多面体的顶点和棱也构成一个图)。我们只关心顶点的多少及这些边是连结哪些顶点的。确切地说，如果两个图 G 与 G' 的顶点之间可以建立起一对一的对应，并且当且仅当 G 的顶点 V_i 与 V_j 之间有 K 条边相连的，G' 的相应的顶点 U'_i 与 U'_j 之间也有 K 条边相连，我们就说 G 与 G' 有相同的结构，简称为同构的。同构的两个图，我们认为是没有区别的。

图 1-4 与图 1-3 乍看起来很不一样，其实这两个图却是相同的，这只要将图 1-4 中的

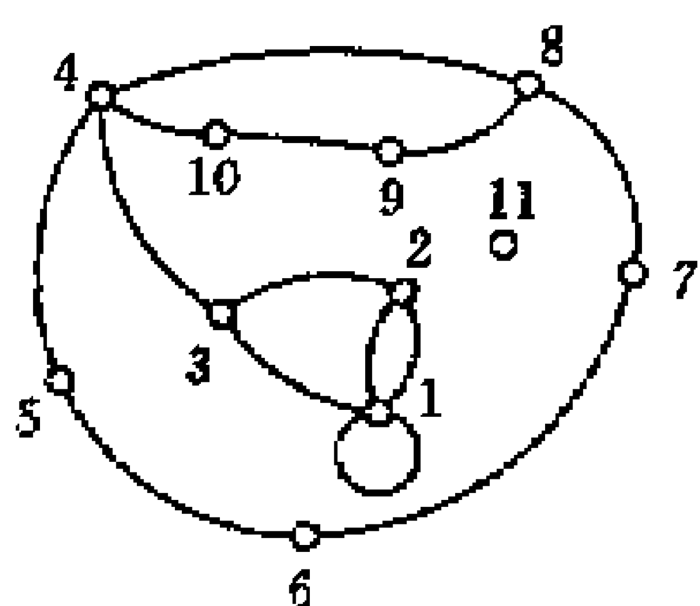


图 1-4

顶点 V_i' 与图 1-3 中的 V_i 相对应（$1\leqslant i\leqslant 11$）就明白了。

通常用一个大写字母 G 来表示图，用 V 来表示所有顶点的集合，E 表示所有边的集合，并且记成 $G=(V,E)$。如果顶点个数 $|V|$ 与边的条数 $|E|$ 都是有限的，图 G 就称为有限图。如果 $|V|=1$，$|E|=0$，图 G 称为平凡图。这种仅含一个孤立点的图是有限图的一种特例。如果 $|V|$ 或 $|E|$ 是无限的，图 G 称为无限图。

本书讨论的图形式都是有限图。

如果对图 $G=(V,E)$ 与 $G'=(V',E')$，G' 的顶点集是 G 的顶点集的一个子集（$V'\subseteq V$），G' 的边集是 G 的边集的一个子集（$E'\subseteq E$），我们说 G' 是 G 的子图。例如，一个正方形就可以看作是图 1-5(*a*)的一个子图，一个五边形也可以看作是图 1-5(*b*)的一个子图。

如果一个图没有环，并且每两个顶点之间最多只有一条边，这样的图称之为简单图。在简单图中，连接 V_i 与 V_j 的边可以记成 (V_i,V_j)。

如果 G 是一个简单图，并且每两个顶点之间都有一条边，我们就称 G 为完全图。通常将具有 n 个顶点的完全图记为 K_n。例如图 1-5(*b*)就是一个完全图 K_5。

如果 G 是一个简单图，它的顶点集合 V 是由两个没有公共元素的子集 $X=\{X_1,X_2,\cdots,X_n\}$ 与 $Y=\{Y_1,Y_2,\cdots,Y_m\}$ 组成的，并且 X_i 与 X_j（$1\leqslant i,j\leqslant n$），$Y_s$ 与 Y_t（$1\leqslant s,t\leqslant m$）之间没有边连接，则 G 叫做二分图。

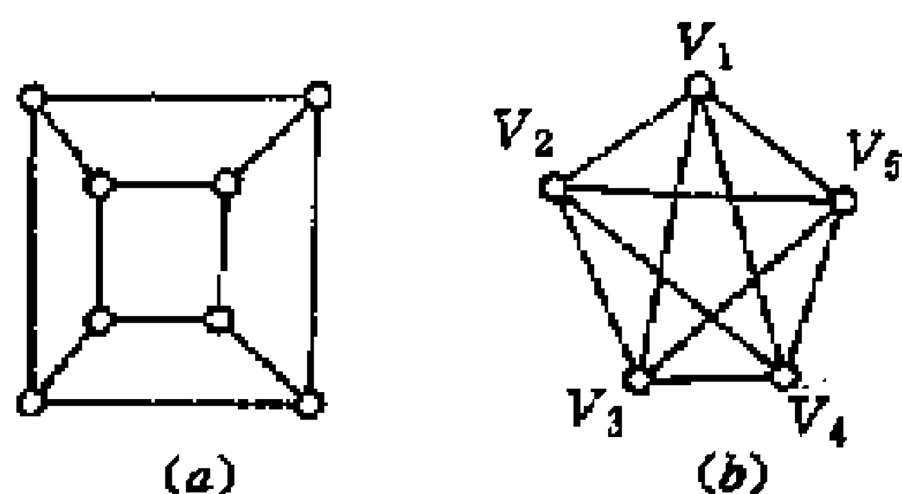


图 1-5

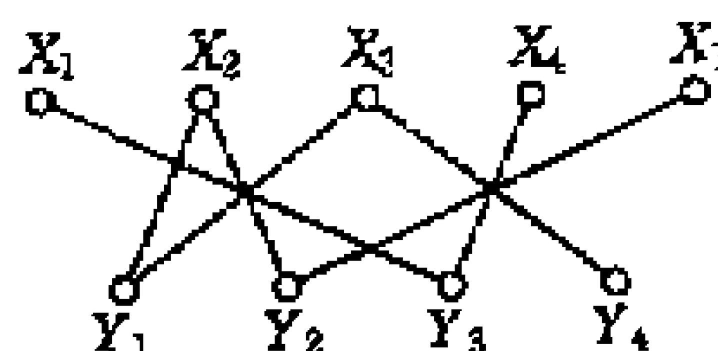


图 1-6

如果在二分图 G 中，$|X|=N$，$|Y|=M$，每一个 $X_i\in X$ 与每一个 $Y_j\in Y$ 有一条边相连，则 G 叫做完全二分图，记为 $K_{n,m}$。显然，图 1-6 为完全二分图 $K_{5,4}$。

如果 G 是一个 N 个顶点的简单图，从完全图 K_n（如图 1-5(*b*)）中把属于 G 的边全部去掉后，得到的图称为 G 的补图，通常记为 $\overline{G}$。例如，图 1-7(*a*)的补图 $\overline{G}$ 为图 1-7(*b*)。

显然 $G=\overline{\overline{G}}$。即一个图的补图的补图就是原来的那个图。

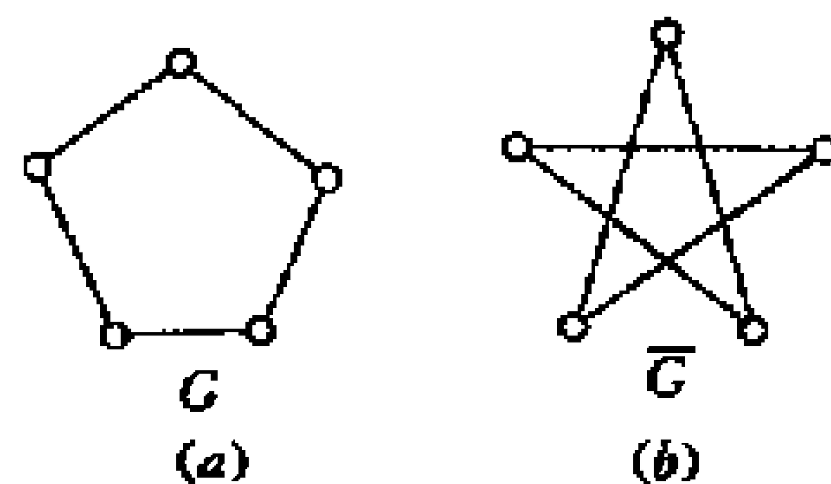


图 1-7

下面我们还要介绍一下相邻与次数这两个术语。

如果图 G 的两个顶点 V_i 与 V_j 之间有边相连，我们就说 V_i 与 V_j 是相邻的，否则就说

V_i 与 V_j 是不相邻的。如果顶点 V 是边 e 的一个端点，就说顶点 V 与边 e 是相邻的，e 是从 V 引出的边。从一个顶点 V 引出的边的条数，称为 V 的度数，记作 $d(V)$。例如图 1-5(b)中，$d(V_1)=d(V_2)=d(V_3)=d(V_4)=d(V_5)=5-1=4$。图 1-6 中的 $d(Y_3)=2$，等等。

以后，我们把每个顶点的次数值为常数 K 的图叫做 K 度正则图。我们也经常使用下面两个符号：

$\& = \min\limits_{1\leqslant i\leqslant |V|}\{d(V_i)\}$，即所有顶点的次数的最小值为 &；

$\Delta = \max\limits_{1\leqslant i\leqslant |V|}\{d(V_i)\}$，即所有顶点的次数的最大值为 Δ。

我们可以从顶点次数问题的讨论中，引出一些有趣的结论：

1. $\sum\limits_{V_i\in V} d(V_i) = 2\times |E|$；

2. 对于任意的图 G，奇次顶点的个数一定是偶数。

这两个结论可以帮助我们分析一些问题。

【例 1】 空间是否有这样的多面体存在，它们有奇数个面，而每个面又有奇数条边？

分析： 构作一个图，以面为顶点，当且仅当两个面有公共棱时，则在 G 的相应两顶点间连一条边，得到图 G。依题意，图的顶点个数是奇数，而且每个顶点的度数 $d(V)$ 是奇数，从而 $\sum\limits_{V_i\in V} d(V_i)$ 也是奇数，与结论 1 相违，故这种多面体不存在。

【例 2】 晚会上大家握手联欢，问是否会出现握过奇次手的人是奇数的情况？

分析： 构作一个图，以人为顶点，两人握手时，则相应的两个顶点之间连一条边，于是每人握手的次数即相应顶点的次数。由结论 2，奇次顶点的个数总是偶数，所以握过奇次手的人数是奇数的情况不可能出现。

1.3 道路与回路

1736 年数学家欧拉(Euler 1707—1783)发表了一篇论文，解决了著名的七桥问题，这一节，我们就来谈谈七桥问题及一些有关内容。

一条河从城市穿过，河中有两个岛 A 与 D，河上有七座桥，连接这两个岛及河的两岸 B，C(图 1-8(a))。

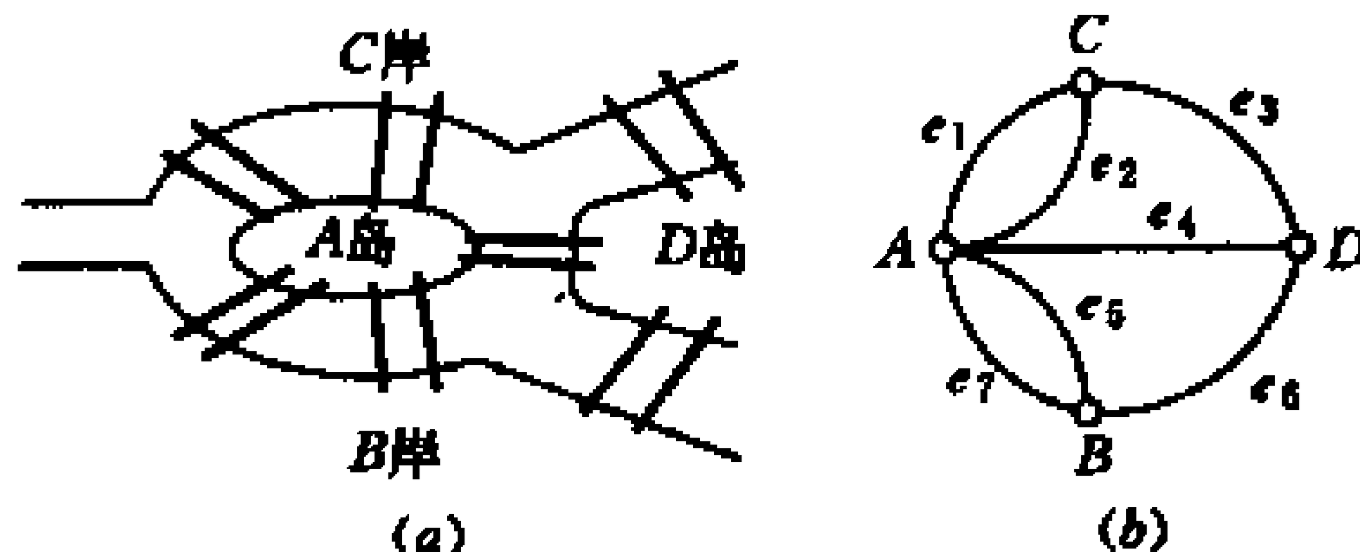


图 1-8

问：

(1) 一个旅行者能否经过每座桥恰好一次，既无重复也无遗漏？

(2) 能否经过每座桥恰好一次，并且最后能够回到原来出发点？

我们把图 1-8(a)改画成如图 1-8(b)所示的图，A,B,C,D 变成四个点，七座桥变成七条边，七桥问题就变成通常所说的一笔画问题：能否一笔画出这个图(每条边都无遗漏，也无重复地画到)？或能否一笔画出这个图、并且最后可以回到原来的出发点？

为了叙述方便及今后的需要，我们再引入几个概念：

在图 G 中，一个由不同的边组成的序列 $e_1,e_2,\cdots,e_g$，如果 e_i 是连接 V_i-1 与 $V_i(i=1,2,\cdots,g)$的，我们就称这个序列为从 V_0 到 V_g 的一条道路，数 g 称为路长，V_0 与 V_g 称为这条道路的两个端点，$V_i(1\leqslant i\leqslant g-1)$叫做道路的内点。如果 G 是简单图，这条道路也可以记作$(V_0,V_1,\cdots,V_g)$。

例如，图 1-9 中，e_1,e_2,e_3,e_4,e_5,e_6 组成一条道路。

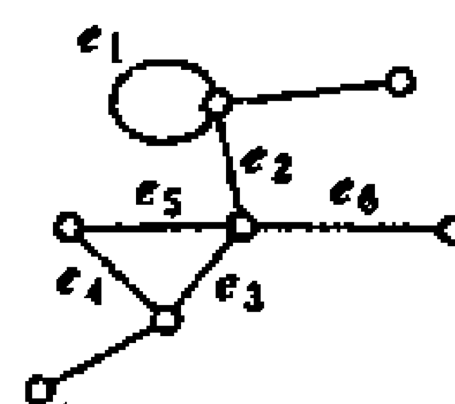


图 1-9

注意在道路的定义中，并不要求 V_0 至 V_g 互不相同。如果 V_0 至 V_g 互不相同，这样的道路称为轨道，记成 $P(V_0,V_g)$。$V_0=V_g$ 的路叫做回路。$V_0=V_g$ 的轨道叫做圈。长为 K 的圈叫做 K 阶圈。不难看出，如果有一条从 V 到 V' 的道路上去掉若干个回路，便可得到一条从 V 到 V' 的轨道。

U,V 两顶点的距离是指 U,V 间最短轨道的长度，记作为 $D(U,V)$。若 U 与 V 之间存在道路，则称 U 与 V 相连通。图 G 中任意两个顶点皆连通时，称 G 为连通图。

利用道路(回路)的概念，图能否一笔画成(并且回到原出发点)的问题，就等价于这个图是不是一条道路(回路)？

如果图 G 是一条从 V_0 到 V_g 的道路，那么该条道路上的每一个内点 $V_i(1\leqslant i\leqslant g-1)$ 都是度数为偶数的顶点。因为对 V_i 来说，有一条进入 V_i 的边，就有一条从 V_i 引出的边，而且进出的边不能重复已走过的边，所以与 V_i 相邻的边总是成双的。故图 G 至多有两个奇顶点，即 V_0 与 V_g。如果 G 是一条回路，那么根据上面推理，V_0 与 V_g 的度数也是偶数。由此，我们可以引出下面一个结论：

有限图 G 是一条道路(即可以一笔画成)的充分必要条件是 G 是连通的，且奇顶点的个数等于 0 或 2，并且当且仅当奇顶点的个数为 0 时，连通图 G 是一条回路(孤立点可以看作是回路)。

显然，由于图 1-8(b)中有 4 个奇顶点，因而不能一笔画成，即一个旅行者要既无重复也无遗漏地走过图 1-8(b)中的七座桥是不可能的。

如果你再深入地探讨一下一笔画的内涵，还可以引伸出下述一些结论，我们可以借助这些结论来分析实际问题：

1. 若连通图 G 有 $2K$ 个奇顶点，那么图 G 可以用 K 笔画成，并且至少用 K 笔才能画成。

例如，图 1-8(b)有 4 个奇顶点，该图可以用 2 笔画成：其中一种方案是：

第一笔画：$B \xrightarrow{e_6} D \xrightarrow{e_4} A \xrightarrow{e_1} C \xrightarrow{e_3} D$

第二笔画：$C \xrightarrow{e_2} A \xrightarrow{e_7} B \xrightarrow{e_5} A$

2. 如果图 G 有两个奇顶点、K 个互相没有公共顶点的连通子图，那么图 G 可以分解成 $K-1$ 条回路和一条道路。

例如图 1-10 中有两个奇顶点 V_2 和 V_5、两个分支(V_1，V_2，V_4，V_5)和 V_3。

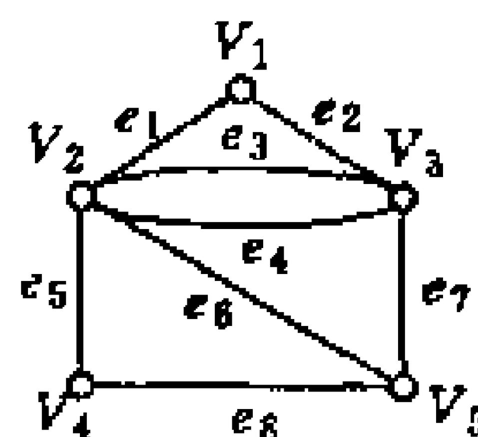


图 1-10

图 G 可以分解成：

1 条回路：(e_1，e_3，e_2)；

1 条道路：(e_8，e_5，e_6，e_7，e_4)。

3. G 为二分图的充分必要条件是 G 中无奇顶点。

例如有一只老鼠在 3×3×3 的乳酪块上咬出一条洞，这个洞通过 1×1×1 的 27 个小立方体的中心，它从大立方体的一角咬起，只要还有它没尝过的小点心块，就继续向前咬。问这只老鼠能否在 3×3×3 立方体中心停止？设这只老鼠是从一个 1×1×1 的小立方体中心沿侧面正交的方向向另一未咬过的小点心块的中心咬去的。

分析：以 1×1×1 的小立方体为顶点构作一个图，把 3×3×3 立方体中心那块小立方体与开始被咬的小立方体之间连一条边，再把有公共侧面的小立方体连上边；以 8 个角上及 6 个侧面中心处的小立方体为 X 集合，其余的小立方体为 Y 集合，于是构成一个二分图。由结论 3，此图不会有 27 阶圈，所以老鼠不会停留在 3×3×3 立方体的中心。

1.4 树

【例 1】 《红楼梦》中荣国府的世系图如下(见图 1-11(a))：

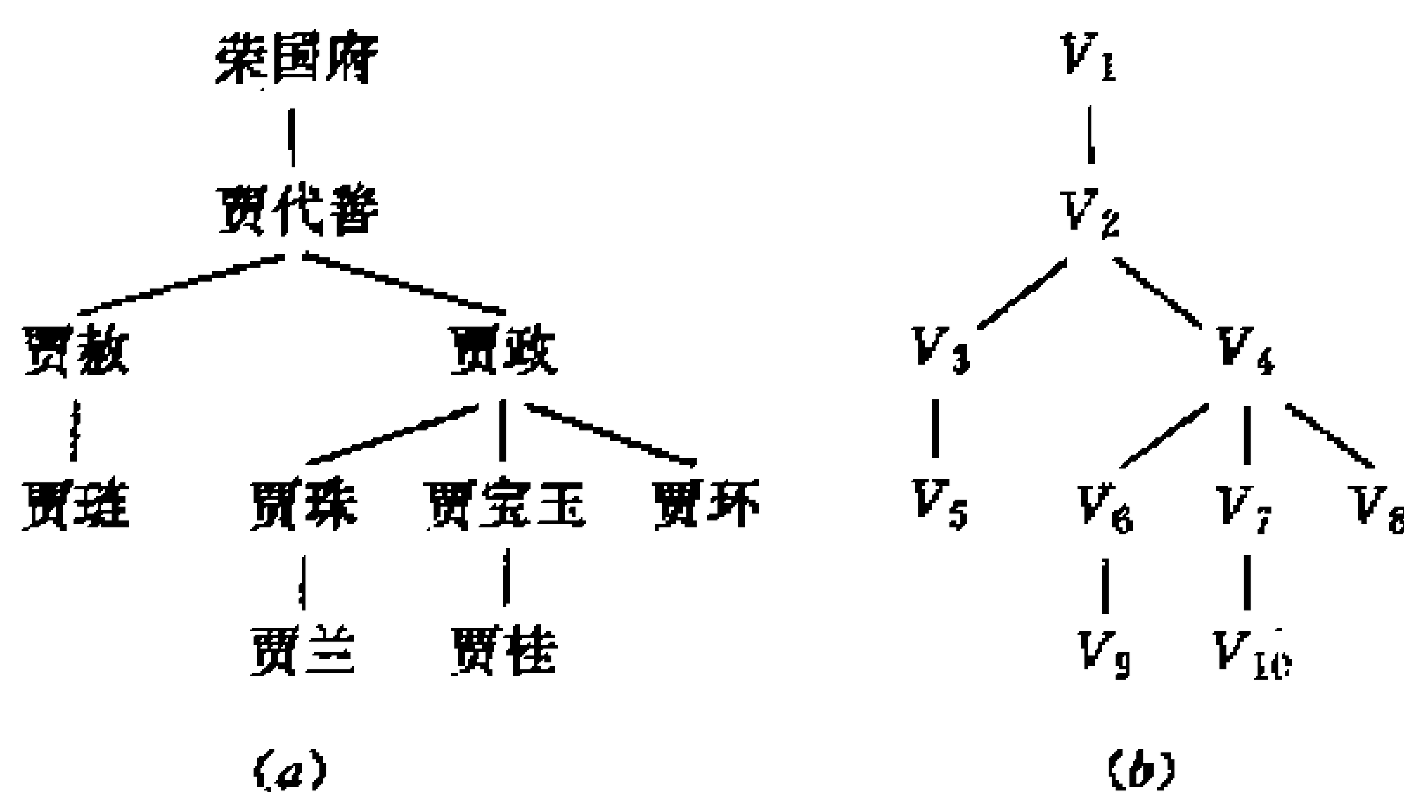


图 1-11

如果将每个人用一个顶点来表示，并且在父子之间连一条边，便得到图 1-11(b)。

这种图称为树，因为它的形状很像一棵倒悬的树。

现在我们给出树的定义：

没有圈的连通图称作树，通常用 T 表示。T 中 $d(V)=1$ 的顶点叫做叶；每个连通分支皆为树的图叫做森林，孤立的顶点叫做平凡树。

图 1-12 是一个森林，每个连通分支皆为树。

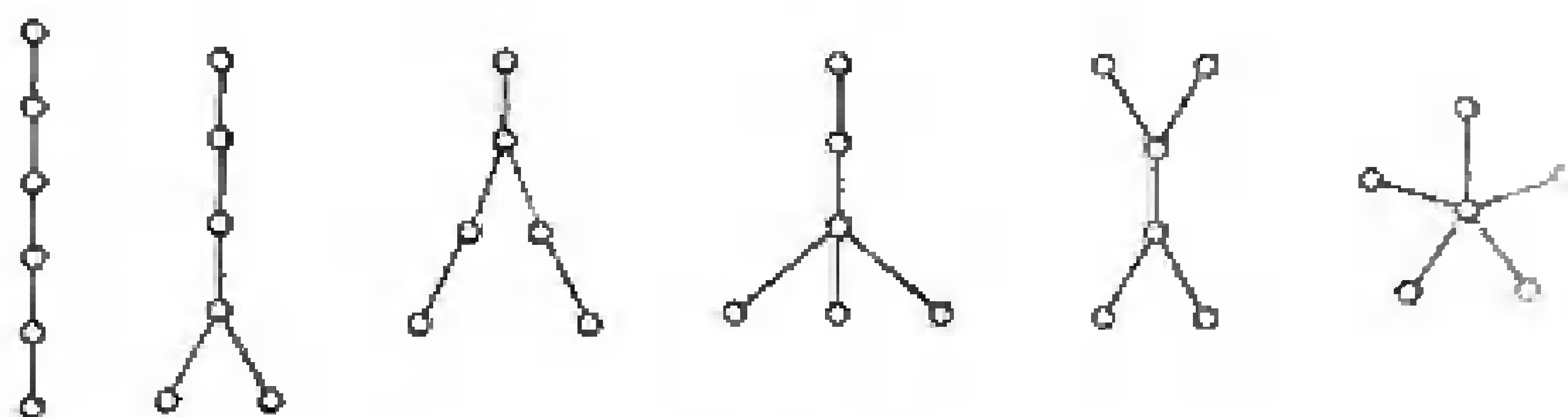


图 1-12

下面，我们通过树的顶点数与边数之间的关系，揭示树的一个图论特征。

如果树 T 的顶点数为 N，那么它的边数 $M=N-1$；倒过来，一个具有 N 个顶点、$M=N-1$ 条边的连通图 G，一定是一棵树。

树 T 具有以下性质：

1. 在 T 中去掉一边后所得的图 G 是不连通的；

2. T 添加一条边后所得的图 G 一定有圈；

3. T 的每一对顶点 V 与 V' 之间有且仅有一条轨道相连。

设 G 是一个连通图，如果 G 中有圈，我们在这个圈中去掉一条边，得到的 G' 还是连通的，如果 G' 仍然有圈，再在圈中去掉一条边得连通图 G''，……，这样继续下去，最后得到一个树 T。T 与 G 的顶点是相同的，并且从 T 陆续添加一些边就得到 G。具有这样性质的树称为连通图 G 的生成树。从 G 中删除 T 的边得到的子图称为 G 的余树。

例如图 1-13 中的粗边便构成该图的一个生成树，而细边便是余树边。可见余树可能不连通。

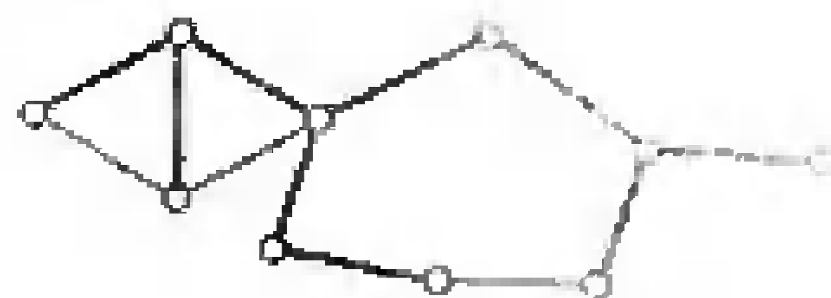


图 1-13

一个具有 n 个顶点的完全图 K_n 可以产生 N^{n-2} 个不同的树。

本章只是图论的一个开头，刚起步就冒出这么多的概念、术语和符号，以后各章节将会讲述这些概念的应用。图论的概念为数甚多。为了加强理解，读者可以参阅图论的其他书籍，但大可不必死记硬背。要多画示意图，从正反两个方面把本质的和易于误解的地方搞清楚，而且要学会用。

第二章　求最短路径的算法及应用

2.1　求最短路

一、什么是最短路问题

这一章先讲一个比较简单但又是很重要的问题——最短路问题。这个问题有着大量的生产实际的背景。事实上大至海陆空各种运输，小至一个人每天上班，都会遇到这一问题，甚至有些问题从表面上看与最短路问题没有什么关系，却也可以归结为最短路问题。下面就举一个这样的例子。

【例1】 渡河问题。一个人带了一只狼、一只羊和一棵白菜想要过河。河上有一只独木船，每次除了人以外，只能带一样东西。另外如果人不在旁时狼就要吃羊，羊就要吃白菜。问应该怎样安排渡河，才能做到既把所有东西都带过河，在河上来回的次数又最少？

我们设变量 M 代表人，W 代表狼，S 代表羊，V 代表白菜，Φ 代表空，什么都没有。开始时设人和其它三样东西在河的左岸，这种情况用 $MWSV$ 表示。

我们用一个集合表示目前左岸的情况。很显然，可能出现的情况有 16 种：

[$MWSV$]，[MWS]，[MWV]，[MSV]，
[WSV]，[MW]，[MS]，[MV]，
[WS]，[WV]，[SV]，[M]，
[W]，[S]，[V]，[Φ]。

剔除下述 6 种可能发生狼吃羊、羊吃白菜的情况：

[WSV]，[MW]，[MV]，[WS]，[SV]，[M]。

现在我们就来构造一个图 G，它的顶点就是剩下的 10 种情况。G 中的边是按下述原则来连的：如果经过一次渡河，情况甲可以变成情况乙，那么就在情况甲与情况乙之间连一条边（见图 2-1）。作了图 G 以后，渡河的问题就归结为下述问题了：在 G 中找一条连接顶点 $MWSV$ 与 Φ，并且包含边数最少的路。如果我们设 G 中各边的长度都是 1，那么也可以把渡河问题归结为："找一条连接 $MWSV$ 与 Φ 的最短路"。把问题转化为图论后，就可用一种系统的方法解决，而不是通常人们所用的凑的方法和凭经验的方法。

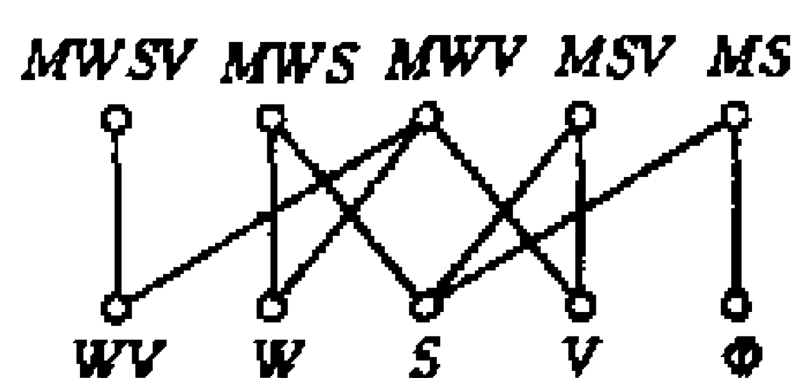


图　2-1

下面，我们可以给最短路问题下一个抽象的定义：

1. 求有向图（图中从一个顶点连到相邻顶点的边有方向性）的最短路问题

设 $G=(V,A)$ 是一个有向图，它的每一条弧 A_i 都有一个非负的长度 $L(A_i)$，在 G 中指定一个顶点 V_s，要求把从 V_s 到 G 的每一个顶点 V_j 的最短有向路找出来（或者指出不存在从 V_s 到 V_j 的有向路，即 V_s 不可达 V_j）。

2. 求无向图(图中连接两个顶点的边无方向性)的最短路问题

设 $G=[V,E]$ 是一个无向图,它的每一条边 e_i 都有一个非负长度 $L(e_i)$。在 G 中指定一个顶点 V_s,要求把从 V_s 到 G 的每一个顶点 V_j 的最短无向路找出来(或者指出不存在从 V_s 到 V_j 的无向路,即 V_s 不可达 V_j)。

二、求最短有向路的标号法

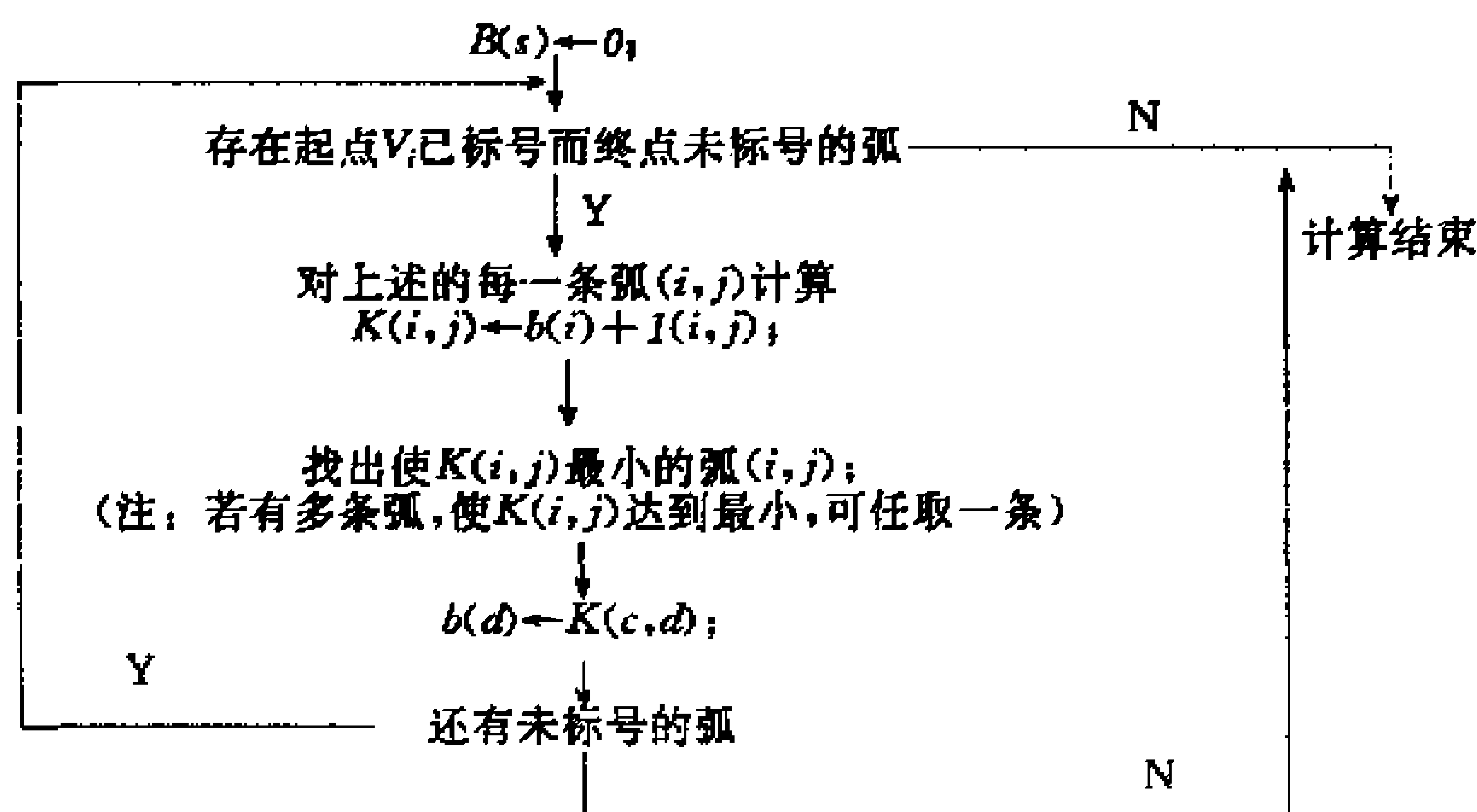
所谓标号,是指与图的每一个顶点对应的一个数字。设:

$b(j)$——顶点 V_j 的标号,代表的是 V_s 到 V_j 的最短的长度。V_j 已标号则意味着 V_s 到 V_j 的最短路以及这条路径的长度已经求出。显然初始时 $b(s)=0$。

$l(i,j)$—弧(V_i,V_j)的非负长度。

$K(i,j)$ —当前有向路加入弧(V_i,V_j)后,V_s 到 V_j 的有向路长度。

标号法的算法流程如下:



由标号法的算法流程可以看出,标号法采用顺推的方法,每边检测一次,没有重复的回溯搜索,因此是一种最佳算法。

注意几个问题:

1. 若是有向图的最短路问题,则用下述两种方法求解:

(1) 将图的每一条边修改为两条方向相反的弧,构成有向图 G'。然后直接用标号法求最短路;

(2) 将标号法略加修改。

① 各个步骤中的弧改为边;② 判断条件"存在起点 V_i 已标号而终点 V_j 未标号的弧"改为"存在一端已标号而另一端未标号的边"就可以直接在无向图上求最短路了。

2. 如果只要求 V_s 到某顶点 V_j 最短路,那么也可以在 V_j 得到标号后就结束计算。

3. 若计算结束,还有一些未标号的顶点,则肯定 V_s 到这些顶点的有向路或无向路不存在。

三、标号法程序

```
program shortest_way;
```

```
  uses crt;
  const  maxn = 100;
  type   list = array[1..maxn] of integer;
  var    chart : array[1..maxn] of list;          { 有向图的邻接矩阵 }
         mark : array[1..maxn] of boolean;
         { 标志表
         { mark[i]= {true   V1 至 Vi 的最短路径已求出}
                    {false  V1 至 Vi 的最短路径未求出 }
         b : list;
         { b[i] ——V1 到 Vi 的最短路径长度 }
         n : integer;                                  { 顶点数 }
procedure init;
  var i,j,a,k : integer;
    begin;
      clrscr;
      repeat write('n='); { 输入顶点数 }
            readln(n);
    until (n>0) and (n<maxn);
    for i :=1 to n do { 邻接矩阵初始化 }
      for j :=1 to n do
        chart[i,j] :=0;
    write('number of lines : '); { 输入边数 }
    readln(a);
    for k :=1 to a do { 输入各边的权 }
      begin
      read(i);
      read(j);
      readln(chart[i,j]);
      end;
    for i :=2 to n do  { 标志表初始化 }
      mark[i] :=false;
    mark[1] :=true;  { 从 V1 顶点出发搜索 }
    b[1] :=0;
  end;
procedure main;
  var   best,                          { 最短路径代价 }
        best_j,                        { 当前的最短路径的端点序号_d}
        i,j : integer;
    begin
      repeat best :=0;
            for i :=1 to n do
          { 从所有起点已标号、终点未标号的弧集中,选一条弧(i,best_j), }
          { 使 V1 至 best_j 的路径长度最短 }
              if mark[i] then
          { V1 至 Vi 顶点的最短路已求出,即起点已标号 }
                for j :=1 to n do
                  if (not mark[j]) and (chart[i,j]>0) then
                    if (best=0) or (b[i]+chart[i,j]<best)
          { 存在一条终点未标号的弧(Vi,Vj)且 Vi 至 Vj 的路目前最短 }
```

```
                    then begin
                      best := b[i]+chart[i,j];
                        { 则记下该路径代价和弧端点 j }
                        best_j := j;
                        end;
          if best>0 then { 若最短路存在 }
            begin
              b[best_j] := best;
                { 记下 V1 至 best_j 的路径代价,并设 best_j 访问标志 }
                mark[best_j] := true;
            end;
      until best=0; { 直至 V1 至其它可以达到的顶点的最佳路径求出 }
    end;
procedure show; { 打印路径代价 }
  var i : integer;
    begin
      for i := 1 to n do
        write(b[i] : 4);
    writeln;
  end;
begin
  init; { 输入图 }
  main; { 求 V1 至其它可以到达的顶点的最短路径 }
  show; { 打印结果 }
end.
```

2.2 服务点设置问题1——求图的中心

一、服务点设置问题的第一个标准

服务点设置问题的一般提法是:设 $G=[V,E]$ 是一个连通的无向图,在它各个顶点 V_i 上,有一些服务对象(例如中小学生)现在要设置一个服务点(例如学校),问服务点设在 G 哪一个顶点上最好?

在研究这类问题时,首先要解决的是好坏标准问题。现在我们设定一个标准——就近入学,就是在来上学的学生中,以最远的学生走的路程为标准。或者说最远的服务对象与服务点的距离应尽可能小。这一节就讨论在这种标准下,如何求最好服务点的问题。

设 $G=[V,E]$ 是一个连通的无向图,每一条边 E_j 有一个非负的长度 $L(E_j)$。现在任取一个顶点 V_i,然后考虑 V_i 与 G 的所有点之间的最短路的长度:

$$d(V_i,V_1),d(V_i,V_2),\cdots,d(V_i,V_n)$$

这 N 个距离中的最大数称为 V_i 的最大服务距离,记做 $e(V_i)$。它的实际意义很清楚:如果把服务点设在 V_i,那么这个服务点与最远的服务对象间的距离至少是 $e(V_i)$。

在所有顶点中,使 $e(V_i)$ 达到最小的顶点叫做图 G 的中心。如果以最大服务距离的大小作为好坏的标准,那么,服务点以设在中心为最好。

二、图中心的计算

要求图的中心,首要的是构造一张 $N\times N$ 的表。图的任意二个顶点之间的最短路集

中写在这表 2-1 上：

表 2-1

	V_1	V_2	…	V_j	…	V_n
V_1	0					
V_2		0				
⋮			⋱			
V_i				$d(i,j)$		
⋮					⋱	
V_n						0

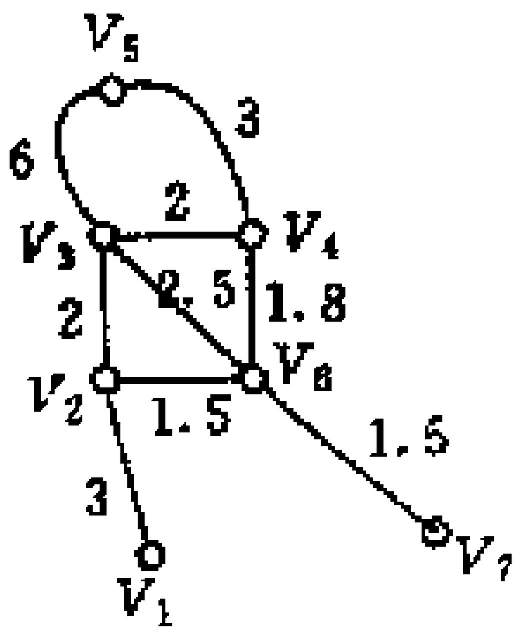


图 2-2

其中第 $i(1\leqslant i\leqslant N)$行表示 V_i 到各个顶点的最短路的长度 $d(i,j)(1\leqslant j\leqslant N)$，注意 $d(i,i)=0$。我们令 $b(i)=0$，用一次标号法将该行上 V_i 至其它 $N-1$ 个顶点的最短路求出，并求出对应的 $e(V_i)$。连算 N 次即可填完图的距离表。

例如：表 2-2 是对应的距离表，$e(V_i)$写在表的最右列。

表 2-2

	V_1	V_2	V_3	V_4	V_5	V_6	V_7	$e(V_i)$
V_1	0	3	5	6.3	9.3	4.5	6	9.3
V_2	3	0	2	3.3	6.3	1.5	3	6.3
V_3	5	2	0	2	5	2.5	4	5
V_4	6.3	3.3	2	0	3	1.8	3.3	6.3
V_5	9.3	6.3	5	3	0	4.8	6.3	9.3
V_6	4.5	1.5	2.5	1.8	4.8	0	1.5	4.8
V_7	6	3	4	3.3	6.3	1.5	0	6.3

要在图 2-2 中选一个服务点，那么从对应的距离表 2-2 可以看出，设在 V_6 最好，因为 $e(V_6)=4.8$ 是所有 $e(V_i)$中最小的。

那么怎样求一个图的中心呢？这个问题其实前面都已经讲了。下面再来总结一下：

步骤 1. 用 2.1 节讲的标号法求出图 G 的距离表；

步骤 2. 求出与一个顶点 V_i 的最大服务距离 $e(V_i)$，$e(V_i)$等于距离表上与 V_i 对应的行中的最大数；

步骤 3. 求出使 $e(V_i)$达到最小的顶点 V_k，V_k 就是图的中心。

由于用一次标号法可以求出一个顶点至其它顶点的最短路长度，因此只要在标号法内增添少许语句，便可求出这个顶点的 $e(V_i)$值，步骤 1，2 实际上是连算几次标号法，剩下的步骤 3 极其简单，只要大约进行 n 次比较大小就可以完成了。大部分工作集中在计算距离表上。标号法求距离表的效率是最为显著的，因此用上面讲的 3 个步骤来求图的中心也是有效的。

三、图中心程序

```
program find_center_point;
uses crt;
const  maxn = 100;                  { 数据结构类同求最短路径 }
type   list = array[1..maxn] of real;
       chart  :  array[1..maxn] of list;
var    mark   :  array[1..maxn] of boolean;
       b      :  list;
       n      :  integer;
procedure init;
  var i,j,a,k : integer;
    begin;
      clrscr;
      repeat write('n=');  { 输入顶点数 }
            readln(n);
      until (n>0) and (n<maxn);
      for i :=1 to n do      { 邻接矩阵初始化 }
        for j :=1 to n do
          chart[i,j] :=0;
      write('number of lines : ');  { 输入边数 a }
      readln(a);
      for k :=1 to a do  { 读入邻接矩阵 }
        begin
          read(i);
          read(j);
          readln(chart[i,j]);
          chart[j,i] :=chart[i,j];
          end;
        end;
function find_farest(x : integer) : real;    {返回 x 顶点的最大服务距离 e(x) }
  var   best_j,i,j             : integer;  { 最短路径的端点序号,辅助变量 }
        best,farest            : real;     { 最短路径代价,最大服务距离 }
    begin
      for i :=1 to n do           { 标志表初始化 }
        mark[i] :=false;
      mark[x] :=true;             { 从 x 顶点开始搜索 }
      b[x] :=0;
      farest :=0;                 { e(x)初始化为 0 }
      repeat best :=0;
            for i :=1 to n do
          { 从所有起点已标号、终点未标号的弧集中,选一条弧(x,best_j), }
          { 使 Vx 至 best_j 的路径长度最短 }
              if mark[i] then { Vi 是起点、已标号 }
                for j :=1 to n do
                  if (not mark[j]) and (chart[i,j]>0) then
                    { 若终点 j 未标号且连接(i,j)后使 Vx 到 Vj 的路径最短 }
                      if (best=0) or (b[i]+chart[i,j]<best)
                    { 则记下路径代价和顶点序号 j,j 结点标号 }
```

```
                then begin
                      best := b[i]+chart[i,j];
                      best_j := j;
                    end;
            if best>0 then
              begin
                b[best_j] := best;
                mark[best_j] := true;
                if best>farest
              { 若 Vx 到 Vbest_j的服务距离目前最大，则记下 }
                  then farest := best;
              end;
      until best=0;
      { 直至 Vx 至其它顶点的最短路径和最大服务距离求出为止 }
      find_farest := farest; { 返回最大服务距离 }
    end;
procedure main;
  var  best_i,i              : integer;
       e,best                : real;
         begin
           best := 0;
             for i := 1 to n do { 求出各顶点的最大服务距离的最小值 }
               begin
                 e := find_farest(i); { 求顶点 i 的最大服务距离 }
                 if (e<best) or (best=0)
         { 若该最大服务距离为目前最小，则记下该顶点序号和它的最大服务距离 }
                   then begin
                          best := e;
                          best_i := i;
                        end;
               end;
             writeln('center point  :  ',best_i); { 打印图的中央点 }
           end;
     begin
         init; { 输入图 }
         main; { 计算和输出图的中心 }
     end.
```

2.3 服务点设置问题 2——求图的 *P* 中心

一、服务点设置问题的第二个标准

仍旧考虑 2.2 节中讲的学校的地点选择问题，不过现在不是设置一所学校，而是设置两所。问设置在什么地方最好？

和前面一样，首先还是要把好坏标准明确一下，仍旧以图 2-2 为例。现在我们先假设学校设在 V_2 和 V_5 两个地方。这时，对于住在 V_1 的学生来说，当他们要读书时，当然到离 V_1 较近的 V_2 去上学，因此 V_1 的学生因读书，至少要走 3 公里的路程，同样的道理，可以

求出 $V_2,V_3,\cdots,V_7$ 的学生上学时必须走的路程(见表 2-3)。

表 2-3

	V_1	V_2	V_3	V_4	V_5	V_6	V_7
V_2	3	0	2	3.3	6.3	1.5	3
V_5	9.3	6.3	5	3	0	4.8	6.3
距离	3	0	2	3	0	1.5	3

从表 2-3 最下面的一行中还可以看出,当学校设在 V_2 和 V_5 时,住在各地的学生上学时需要走的路最多不超过 3 公里。我们把这种情况记作 $e(V_2,V_5)=3$。$e(V_2,V_5)$称作顶点对 V_2,V_5 的最大服务距离。

又例如从表 2.4 可以看出,顶点对 V_1,V_7 的最大服务距离是 6.3 ,即 $e(V_1,V_7)=6.3$,和 $e(V_2,V_5)=3$ 一比较,就可以看出设置在 V_2、V_5 比设置在 V_1、V_7 好。

表 2-4

	V_1	V_2	V_3	V_4	V_5	V_6	V_7
V_1	0	3	5	6.3	9.3	4.5	6
V_7	6	3	4	3.3	6.3	1.5	0
距离	0	3	4	3.3	6.3	1.5	0

讲到这里,好坏标准问题就不难解决了,我们可以规定顶点对最大服务距离愈小愈好,使 $e(V_i,V_j)$达到最小的顶点对称为图的 2 中心。因此,要研究哪两个顶点设置学校最好,就归结为求图的 2 中心了。

图的 2 中心这个概念还可以推广,即可以考虑图的 3 中心,图的 4 中心,……或者一般地说可以考虑图的 P 中心。在研究如何设置 P 个学校最好时,就会遇到求图的 P 中心的问题。

至于规定在 P 中心设置学校的实际意义和 2.2 节中是一样的,这里不再赘述。

二、图的 *P* 中心的计算

如何求图的 2 中心,最简单的方法就是:

1. 用标号法求距离表;
2. 对照距离表将所有顶点对 V_i、V_j 对应的最大服务距离 $e(V_i,V_j)$都找出来;
3. 求使 $e(V_i,V_j)$达到最小的顶点对。

对于一个有 N 个顶点的图来说,在用标号法求出距离表后,还要计算 $n(n-1)/2$ 个顶点对的最大服务距离,而每一次求最大服务距离只要进行约 $3N$ 次比较大小的运算,因此总起来说,上述算法还是有效的。

我们还可以用相似的方法求图的 P 中心:

1. 用标号法求距离表;

2. 对照距离表求所在 P 个顶点的最大服务距离 $e(V_{i1},V_{i2},\cdots,V_{iP})$；

3. 求使 $e(V_{i1},V_{i2},\cdots,V_{iP})$ 达到最小的 P 个顶点。

求图的 P 中心的算法不是一个有效的算法。因为 P 顶点的最大服务距离有 $G_n^P = n!/((n-P)!\times P!)$，而每次 $e(V_{i1},V_{i2},\cdots,V_{iP})$ 要进行约 $(P+1)*N$ 次比较大小的运算。因此计算步数是一个依赖于顶点数 N 和 P 的指数函数。图的 P 中心的算法效率至今还是一个悬而未决的难题。

三、图的 P 中心的程序

```
program p_zhong_xin;
  uses crt;
  const  maxn = 50;
  type   list = array [1..maxn] of real;
  var    chart        : array[1..maxn] of list;    { 图的邻接矩阵 }
         biao         : array[1..maxn] of list;    { 距离表 }
         mark         : array[1..maxn] of boolean;{ 标号表 }
         s,b_s,             { s——组合方案,b_s——目前最好的组合 }
         way,b_way    : array[1..maxn] of integer;
    { way——辅助变量 }
    { b_way——最佳方案,即服务点 i 选择 p 中心中最近的服务点 b_s[b_way[i]] }
         n,p          : integer;                 { n——顶点数, p——中心数 }
         bst          : real;            { 目前最大服务距离的最小值 }
procedure init;
  var i,j,a,k : integer;
    begin;
      clrscr;
      repeat write('n=');                          { 输入顶点数 }
             readln(n);
      until (n>0) and (n<maxn);
      for i :=1 to n do
        for j :=1 to n do
          chart[i,j] :=0;
      write('number of lines : ');                 { 输入边数 }
      readln(a);
      for k :=1 to a do                            { 输入各条边的长度 }
        begin
          read(i);
          read(j);
          readln(chart[i,j]);                      { 各条边的长度 }
          chart[j,i] :=chart[i,j];
        end;
      repeat write('p=');
             readln(p);                            { 输入中心数 p }
      until (p>0) and (p<=maxn);
    end;
procedure make_one_line(x : integer); { 构造距离表 biao 中的 x 行各元素 }
```

```
  var  best_j,i,j  :  integer;
        best        :  real;
    begin
      for i:=1 to n do { 标号表初始代 }
        mark[i]:=false;
      mark[x]:=true;
      biao[x][x]:=0;
      repeat  best:=0;
              for i:=1 to n do
                if mark[i] then
                  for j:=1 to n do
                    if (not mark[j]) and (chart[i,j]>0) then
                      if (best=0) or (biao[x][i]+chart[i,j]<best)
                        then begin
                               best:=biao[x][i]+chart[i,j];
                               best_j:=j;
                             end;
              if best>0 then
                begin
                  biao[x][best_j]:=best;
                  mark[best_j]:=true;
                end;
      until best=0;
    end;
procedure make_biao;                          { 构造距离表 }
  var i  :  integer;
    begin
      for i:=1 to n do
        make_one_line(i);
      end;
procedure check;
  var  i,j,k  :  integer;
        t,v    :  real;
    begin
      v:=0;                                   { v 为当前方案的最大服务距离 }
      for i:=1 to n do
        begin
          t:=biao[s[1]][i];
          k:=1;
          for j:=2 to p do
      { 在当前 p 个结点中,选择离 vi 较近的一个结点 vk,vk 至 vi 的最短距离 }
      { 为 t(vk 为 p 组合的第 k 个元素) }
            if biao[s[j]][i]<t
            then begin
                   t:=biao[s[j]][i];
                   k:=j;
                 end;
```

```
        if t>v
          then v := t;
        way[i] := k; { 存储 vi 至组合方案的 p 个服务点中最近的一个服务点 }
      end;
    if (v<bst) or (bst=-1)          { v 为目前最小的最大服务距离 }
      then begin
             bst := v;
             for i := 1 to p do         { 存储各顶点最近的服务点 }
               b_s[i] := s[i];
             for i := 1 to n do
               b_way[i] := way[i];
           end;
    end;
procedure main;
  var lev : integer;
    begin
      bst := -1;
      lev := 1; s[1] := 0;      { s 中是 p 个服务点的组合方案 }
      while lev>0 do
      begin
        while s[lev]<n do
          begin
            inc(s[lev]);
            if lev=p
      { 若产生一个 p 组合，则求当前 p 组合的最大服务距离，并且得出目前 }
      { 扩展出来的所有 p 组合中最小的最大服务距离方案 }
              then check
              else begin
                     inc(lev);
                     s[lev] := s[lev-1];
                   end;
            end;
          dec(lev);
        end;
      end;
procedure show;
  var i : integer;
    begin
      writeln('you should select : ');
      for i := 1 to p do           { 打印 p 个服务点 }
          write(b_s[i] : 4);
        writeln;
        for i := 1 to n do         { 打印各顶点去哪个服务点的信息 }
          writeln('people in ',i,' go to hospital in ',b_s[b_way[i]]);
    end;
begin
  init;                 { 输入图 }
  make_biao;            { 构造距离表 }
  main;                 { 计算图的 P 中心 }
  show;                 { 输出 P 个服务点以用各顶点去哪个服务点的信息 }
end.
```

2.4 服务点设置问题3——求图的中央点

一、服务点设置问题的第三个标准

再考虑一个服务点设置问题的例子。

我们再把图 2-3 看成一个矿区，它有 7 个矿，分别在 $V_1, V_2, \cdots, V_7$ 处，这 7 个矿每天的矿产量分别是 V_1: 3000t，V_2: 2000t，V_3: 7000t，V_4: 1000t，V_5: 5000t，V_6: 1000t，V_7: 4000t（见图 2-3）。

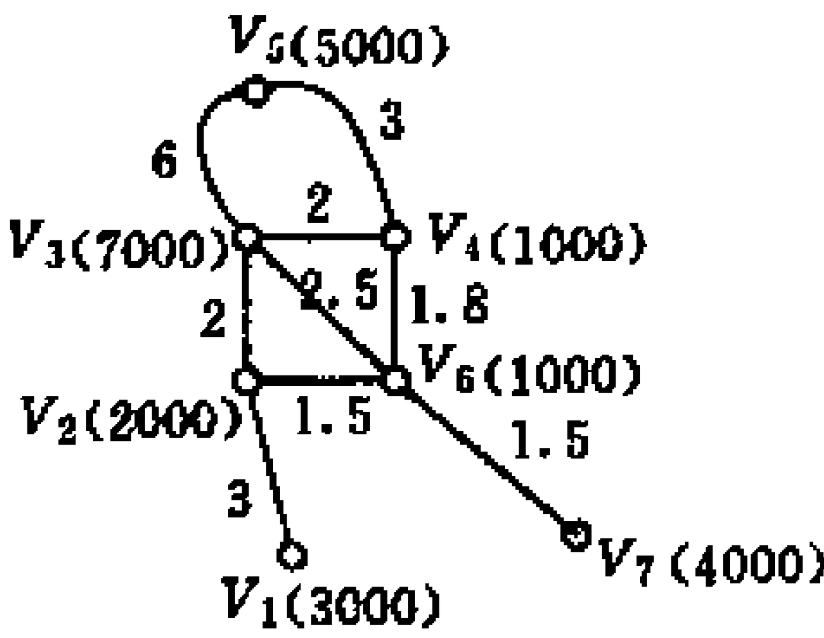


图 2-3

现在要从 $V_1, V_2, \cdots, V_7$ 中选一个地点来建选矿厂，问应该选哪一个地方，才能使各矿生产的矿石往选矿厂运时花费的运输力量最小？

这个问题中，判断一个顶点好坏的标准不同于求图的中心和 P 中心。对一个顶点 V_i，这里应该计算的是，如果选矿厂设在 V_i，那么，将各个矿生产的矿石都运到 V_i 的运输量是多少？然后再来比较在哪里设选矿厂最好。一般，我们以运输的 t·km 数来计算运输量的大小。一吨货物运输一公里就叫 1t·km。如果选矿厂设在 V_1，则运输矿石共需：

$g(V_1)=3000\times0+2000\times3+7000\times5+1000\times6.3+5000\times9.3+1000\times4.5+4000\times6=122300(\text{t}\cdot\text{km})$

$g(V_2)=3000\times3+2000\times0+7000\times2+1000\times3.3+5000\times6.3+1000\times1.5+3000\times3=68300(\text{t}\cdot\text{km})$

一比较，就可以看出选矿厂设在 V_2 要比设在 V_1 好。当然 V_2 是不是最好的还不能确定，应该把 $g(V_1), g(V_2), \cdots, g(V_7)$ 都算出来再比较一下，才可以选出一个最好的设厂点来。

表 2-5 给出了 $g(V_i)$，$i=1,2,\cdots,7$，从表中可见 V_3 最好。

表 2-5

	V_1	V_2	V_3	V_4	V_5	V_6	V_7	$G(V_i)$ (kt·km)
V_1	0	3	5	6.3	9.3	4.5	6	122.3
V_2	3	0	2	3.3	6.3	1.5	3	68.3
V_3	5	2	0	2	5	2.5	4	64.5
V_4	6.3	3.3	2	0	3	1.8	3.3	69.5
V_5	9.3	6.3	5	3	0	4.8	6.3	108.5
V_6	4.5	1.5	2.5	1.8	4.8	0	1.5	65.8
V_7	6	3	4	3.3	6.3	1.5	0	88.3
产量(kt)	3	2	7	1	5	1	4	

上面我们虽然只通过一个例子来讲，不过一般的情况也是这样的。一般说来，设有一个连通图 $G=[V,E]$，G 的每一条边 e_j 有非负的长度 $L(e_j)$，G 的每一个顶点 V_i 还有一个“产量”$A(V_i)$。对于每一个顶点 V_i 令：

$$g(V_i)=A(V_1)\times D(V_i,V_1)+A(V_2)\times D(V_i,V_2)+\cdots+A(V_n)\times D(V_i,V_n)$$

$g(V_i)$代表把各个点的物资运到 V_i 所花费的 t · km 数。对于一个具有 N 个顶点的图 G 来说，使 $g(V_i)$最小的顶点为该图的中央点。在以运输量的大小为标准时，中央点就是设置服务点的最好位置了。

二、图的中央点的计算

求图的中央点的方法是：

1. 用标号法求距离表；
2. 对照距离表求 $g(V_1),g(V_2),\cdots,g(V_n)$；
3. 求使 $g(V_i)$最小的顶点 $V_k(1\leqslant i\leqslant n)$。

由上述算法可以看出，求中央点的计算方法也是有效的。

三、求中央点程序

```
program find_zhong_yang_point;
  uses crt;
  const maxn = 100;
  type  list =  array[1..maxn] of real;          {                    }
  var   chart   : array[1..maxn] of list;         {                    }
        mark    : array[1..maxn] of boolean;      {   类同于 P 中心    }
        b       : list;                           {                    }
        n       : integer;                        {                    }
        aa      : list;                           {  日矿产量表        }
procedure init;
  var i,j,a,k : integer;
    begin;
      clrscr;
      repeat write('n=');      { 输入顶点数 }
            readln(n);
       until (n>0) and (n<maxn);
       for i :=1 to n do      { 邻接矩阵初始化 }
         for j :=1 to n do
           chart[i,j] :=0;
       write('number of lines : ');  { 输入边数 }
       readln(a);
       writeln('input graph : ');
       for k :=1 to a do          { 输入无向图的邻接矩阵 }
         begin
           read(i);
           read(j);
           readln(chart[i,j]);
           chart[j,i] :=chart[i,j];
```

```
      end;
    writeln('input a:');        { 输入日矿产量 aa[1..n] }
    for i:=1 to n do
      readln(aa[i]);
  end;
function find_farest(x:integer):real;
{ 求各顶点物资运到 x 所花费的 t·km 数}
  var best_j,i,j          : integer;  { 最佳路径的端点序号,辅助变量 }
      best,farest         : real;     { 最佳路径代价,g(x)值          }
  begin
    for i:=1 to n do
      mark[i]:=false;
    mark[x]:=true;
    b[x]:=0;
    farest:=0;
    repeat best:=0;
           for i:=1 to n do
             if mark[i] then { 若 Vx 与 Vi 的最短距离已求出 }
               for j:=1 to n do
                 if (not mark[j]) and (chart[i,j]>0) then
                   if (best=0) or (b[i]+chart[i,j]<best)
                     then begin
                            best:=b[i]+chart[i,j];
                            best_j:=j;
                          end;
           if best>0 then{ 若 best_j 顶点与 x 顶点的最短距离求出 }
             begin
               b[best_j]:=best;{ 则存储该距离和 best_j 顶点标号 }
               mark[best_j]:=true;
               farest:=farest+best*aa[best_j];{ 累计 g(x) }
             end;
    until best=0;
    find_farest:=farest;
  end;
procedure main;
  var best_i,i         : integer;  { g 值最小的顶点序号,辅助变量 }
      e,best           : real;     { 顶点 i 的 g 值,目前最小的 g 值   }
  begin
    best:=0;
    for i:=1 to n do { 求 g 值最小的顶点 best_i }
      begin
        e:=find_farest(i);{ 求 g[vi] }
        if (e<best) or (best=0)
          then begin
                 best:=e;
                 best_i:=i;
               end;
      end;
    writeln('center point : ',best_i);{ 打印图的中央点 }
```

```
    end;
begin
  init; { 输入图 }
  main; { 计算并输出图的中央点 }
end.
```

第三章　求最小生成树

3.1　求无向图的最小生成树

这一章讲的最小生成树问题，是图论中又一个很重要的问题，其重要性不亚于最短路问题。先讲一讲什么叫生成树。

一、最小生成树的由来

设 $G=[V,E]$ 是一个无向图，如果 $T=[V,E_1]$ 是由 G 的全部顶点及一部分边组成的子图并且 T 是树（连通、没有圈的图），则称 T 是 G 的一个生成树。

一个连通图 G 一般有许多种生成树。现在考虑一个连通图 $G=[V,E]$，它的每一条边 E_j 有一个长度 $L(E_j)>0$。这时对于 G 的任意一个生成树 T，我们把属于 T 的各条边的长度加起来的和称为 T 的长度，记作 $L(T)$。例如对图 3-1 中的图 G 来说，(b)、(c)中的两个生成树 T_1 与 T_2 的长度就分别为 $L(T_1)=22$，$L(T_2)=17$。

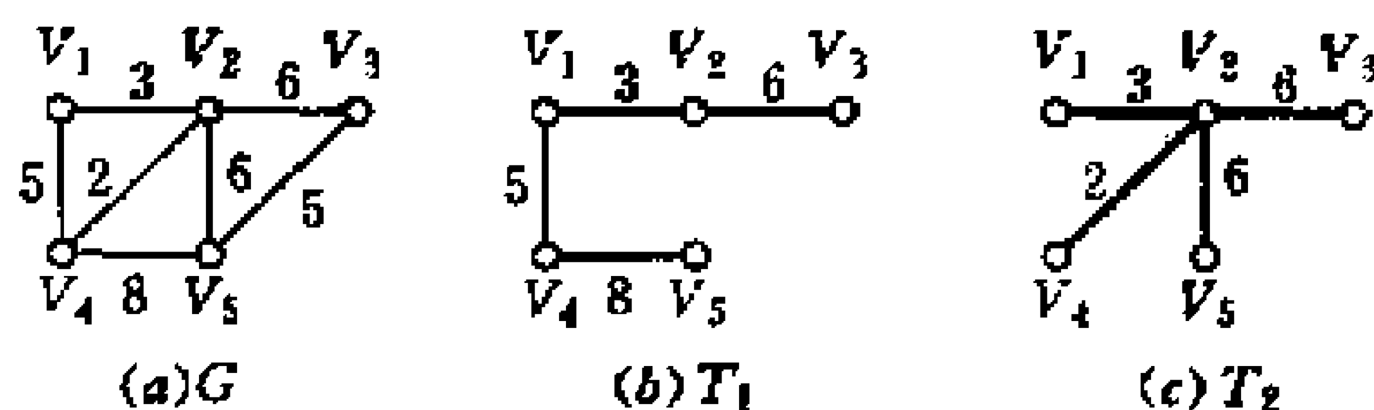


图　3-1

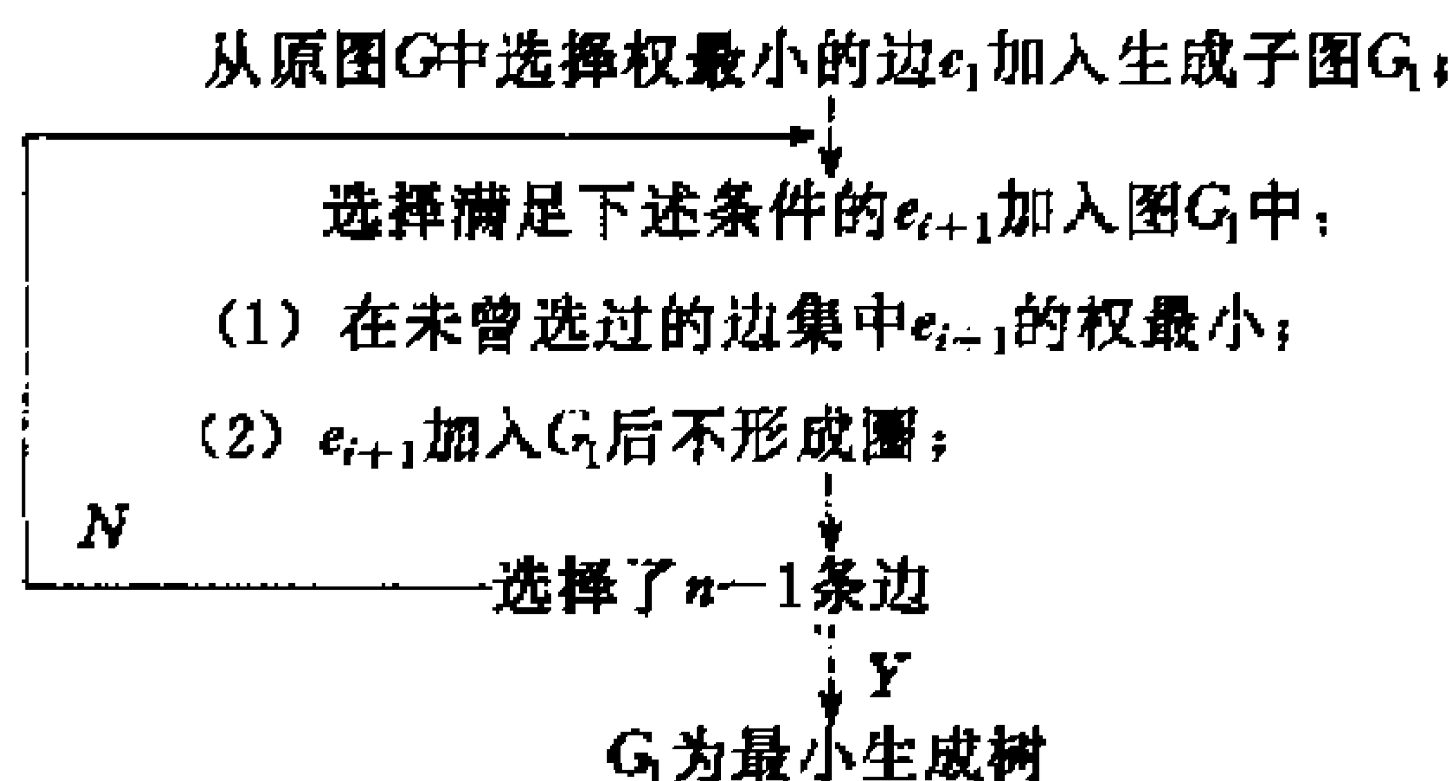
现在可以提出一个问题：如何从 G 的所有生成树中，找出长度最小的生成树。这个问题即所谓最小生成树问题。

最小生成树问题有很广泛的应用。例如，我们把图 3-1 中图 G 中的五个顶点看成某乡的 5 个村，G 的边看成是公路，现在要沿公路架设电线，使各村之间都通电话，问应该怎样架线，才能使所用的电线最少？

考虑一下就可以看出，这个问题的关键是决定图上的哪些边上该架线，哪些边上不架线。设架线的边的集合是 E_1，那么 $G_1=[V,E_1]$ 就是 G 的一个子图。因为架线后各个乡间都能通话，所以 G_1 必须是连通的。因此要使电线最节约，就是要从 G 的所有连通的子图中，把总边长最小的找出来。但是不难看出，总边长最小的连通子图一定不会含圈，从而必定是最小生成树。因此架设电线的问题，就可以归结为最小生成树问题。类似的问题还有很多，例如修公路把一些城镇连接起来，修渠道使水渠和各块地连接起来……，都可以归结为最小生成树问题。另外，有不少图论问题在计算时，往往首先必须求出一个最小生成树问题，这也是最小生成树问题显得特别重要的一个原因。

二、最小生成树的计算

一开始，先将 G 图中的边都去掉，只留下孤立的顶点，这个图即为 G 图最初的生成子图 G_1。然后逐步地将当前最小边 e_1 加上去，每次加的时候，要保持住"没有圈"这一性质，在加了 $N-1$ 条边(N 是顶点个数)后，G_1 便成为所要求的最小生成树了。其计算步骤如下：



三、最小生成树的程序

```
program kruskal_p33;
  const
    maxn          =  30;
  type
    ghtype       =  array [1..maxn,1..maxn] of integer;
    settype      =  set of 1..maxn;
  var
    n            :  integer; { 顶点数 }
    gh           :  ghtype;  { 邻接矩阵 }
    f            :  text;    { 文件变量 }
  procedure read_graph;
    var
      str : string;
      i,j : integer;
    begin
      write('Graph file = '); { 将读入的外部文件名与文件变量连接起来 }
      readln(str);
      assign(f,str);
      reset(f);      { 读准备 }
      readln(f,n);  { 读入顶结点数 }
      for i : =1 to n do
        for j : =1 to n do read(f,gh[i,j]); { 读入邻接矩阵 }
      close(f) { 关闭文件 }
    end;
  procedure select(var a,b : integer;s : settype);
    var
      i,j,t : integer;
```

```
  begin
    t : =maxint;
    for i : =1 to n do
  { 输入当前最小生成树 T 中的顶点集合,在所有待扩展顶点中选择权最小的 }
  { 边<i,j>(i∈T,j∉T)扩展 }
      if i in s
        then for j : =1 to n do
              if (i<>j) and not (j in s) and (gh[i,j]<t)
                then begin
                      a : =i;b : =j;t : =gh[i,j]
                    end
  end;
procedure kruskal;
  var
    i,j,k,t : integer;
    s : settype;
  begin
    t : =0;s : =[1]; { 从结点 1 开始扩展 }
    for i : =1 to n-1 do { 逐一扩展 n-1 条边 }
      begin
        select(j,k,s); { 选边扩展 }
        t : =t+gh[j,k];  { 求权和 }
        writeln(j,'——',k); { 打印该边 }
        s : =s+[k];            { 扩展后的顶点进入 T 树 }
      end;
    writeln('Tot = ',t)
  end;

begin
  read_graph; { 输入图 }
  kruskal      { 计算和输出图的最小生成树 }
end.
```

3.2 求有向图的最小树形图

一、什么叫最小树形图

3.1 节讲最小生成树时,曾说过,最小生成树问题也可以用来解决渠道设计问题。但这种说法仅适用于水源以及需要灌溉的各块地的地势一样高的情况,如果水源和各块地的地势有高低,问题就比较复杂了。

例如图 3-2(*a*)的有向图 D 中 V_1 表示处于最高地势的水源,其它顶点表示需要灌溉的地,各条弧代表可以选来修建渠道的线路,弧的方向指出地势的高低。例如 V_3 到 V_5 有弧表示水可以从高地势的 V_3 流到低地势的 V_5。现在需要解决的问题是:应该选择哪几条路线修建渠道才最节省?

如果不考虑弧的方向,用 3.1 节的办法求最小生成树,那么求得的将是(*b*),它的弧总

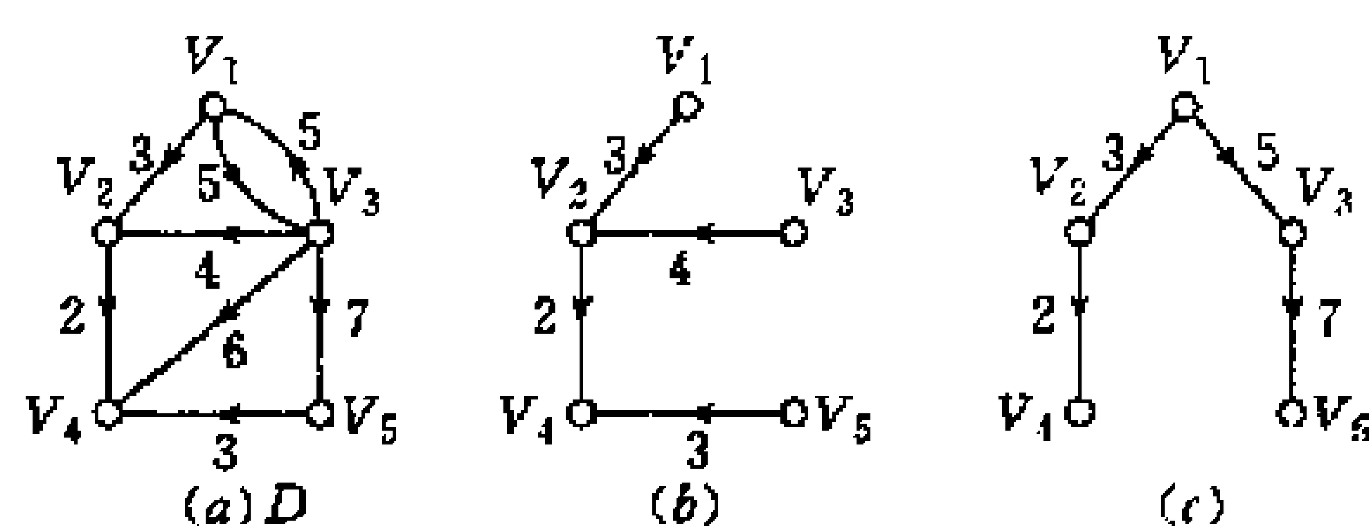


图 3-2

长是 12。但是按这个图修渠道是行不通的，因为 V_1 的水流不到 V_3 和 V_5。要使各块地都能得到灌溉而且总长度又最小的设计方案应该是(c)中画的那样，虽然它的弧总长是 17，但是它是 V_1 的水能够流到各块地的最小值。

分析一下图 3-2(b)和(c)有什么特点。显然它们的公共特点是，如果不考虑弧的方向，它们都是树，即它们的相伴无向图是树。但是作为一个渠道设计的方案来说，单单具有这个性质还不够，它至少还需要一个性质，就是：水源的水能流到各块地上去。

从上述分析中，我们抽象出树形图的定义。

设 $D=(V,A)$ 是一个有向图，如果它具有下述性质：

1. D 不包含有向圈；

2. 存在一个顶点 V_i，它不是任何弧的终点，而 V 的其它顶点都恰好是唯一的一条弧的终点。

则称 D 是以 V_i 为根的树形图。

显然(c)中的图代表的渠道设计方案所以是可行的，原因就在于它是一个树形图。而(b)所以不可行也就是因为它不是树形图。

由树形图的定义，不难引出最小树形图的概念：

设给定了有向图 $D=(V,A)$，它的每条弧都有一个非负的长度，现在要从 D 的所有以 V_i(相当水源)为根的树形图中，找出弧的总长度最小的树形图来。

显然选择最节约的渠道设计方案问题可以归结为求最小树形图问题。

二、最小树形图的求法

求最小树形图要比求最小生成树麻烦一些，不过只要稍为耐心些学习，还是能学懂的。为确定起见，下面都假设指定为根的顶点是 V_1。

步骤 1. 求最短弧集合 A_0

从所有以 V_2 为终点的弧中取一条最短的，再从以 V_3 为终点的弧中取一条最短的……。若在选取以 $V_2,V_3,\cdots,V_n$ 为终点的最短弧的过程中发现一个顶点 $V_j(V_j\neq V_1)$ 不是图 D 中任何弧的终点，这时计算结束，因为显然 D 中不存在以 V_1 为根的最小树形图。若得到以 $V_2,V_3,\cdots,V_n$ 为终点的 $N-1$ 条最短弧，把这些弧组成的集合叫做 $A_0(V,A_0)$ 是图 D 的一个生成子图，A_0 是所有具备了树形图性质 2 的生成子图中弧总长度最短的一个。但是 A_0 是否一定具备了树形图性质 1、一定为 D 的最小树形图呢？

例如对于图 3-3(a)中的有向图 D 来说，用上述办法得到的生成子图 A_0 就是(b)中的

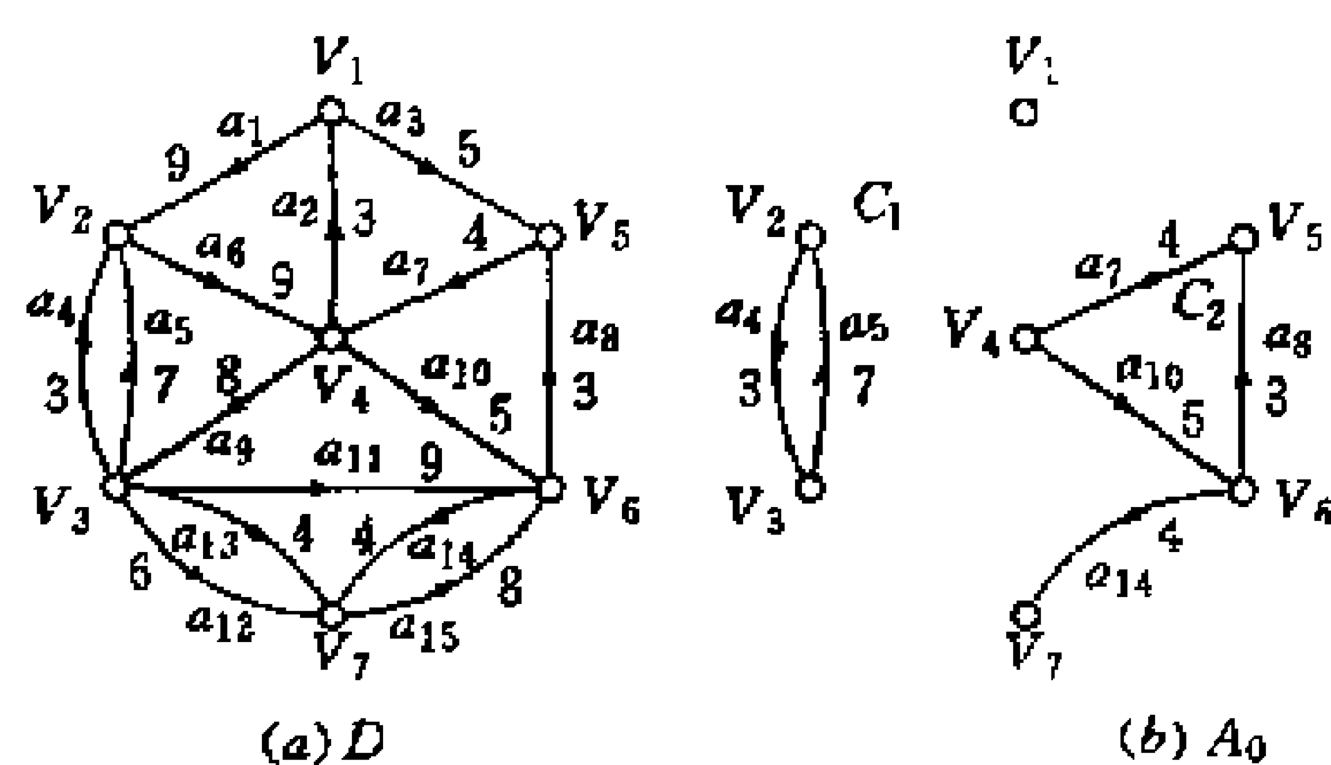


图 3-3

图，它有两个有向圈 C_1 和 C_2，因此 A_0 不是 D 的树形图。

步骤 2. 检查 A_0

若 A_0 没有有向圈且不含收缩点，则计算结束，A_0 就是 D 的以 V_1 为根的最小树形图 H；若 A_0 没有有向圈、但含收缩点，则转步骤 4，若 A_0 含有有向圈 C_1、C_2，…，C_s，转入步骤 3。

步骤 3. 收缩 D(见图 3.3(a))中的有向圈

把 D 中的 C_1,C_2,…,C_s 分别收缩成顶点 U_1,U_2,…,U_s，D 中两个端点都属于同一个有向圈 C_i 的弧都被收缩掉了，其它弧仍保留，得到一个新的图 D_1。D_1 中以收缩点为终点的弧 A_k 的长度要变化，变化的规律是：设 A_k 的终点是收缩点 U_i，并且有向圈 C_i 中，与 A_k 有相同终点的弧 A_t，则

A_k 在 D_1 中的长度 $=L(A_k)-L(A_t)$

这里 $L(A_k)$与 $L(A_t)$分别表示弧 A_k 与 A_t 在 D 中的长度。

按上述收缩的规律，图 3-3 中 D 应该变成图 3-4 中的 D_1。

即把 D 中有向圈 C_1 和 C_2 见图 3-3(b)分别收缩成 D_1 中的顶点 U_1 和 U_2，U_1 和 U_2 今后称为收缩点。

那么、D_0 中有向圈 C_1,C_2,…,C_s 被收缩后得到的 D_1，与原来的 D_0 有什么关系呢？

1. 如果 D_1 没有以 V_1 为根的树形图，那么 D_0 也没有以 V_1 为根的树形图；

2. 如果 H_1 是 D_1 的以 V_1 为根的树形图，那么可以通过某种"展开"的方法(由步骤 4 给出)得到 D_0 的以 V_1 为根的树形图 H_0。

因此，必须返回步骤 1，求 D_1 的以 V_1 为根的树形图。

步骤 4. 展开收缩点

对图 3-4 求最短弧集合 A_1 的结果如图 3-5(a)所示。

由于 A_1 没有有向圈，显然 A_1 就是 D_1 的以 V_1 为根的最小树形图 H_1。但 H_1 含收缩点 U_1 和 U_2。如何将它们"展开"，求得 D_0 的以 V_1 为根的最小树形图呢？展开办法：

所有 D_1 中属于 H_1 的弧在 D_0 中都仍属于 H_0；将每一个收缩点 U_i 展开成有向圈 C_i，C_i 中除去一条与 H_1 中的弧有相同终点的弧外，其它的弧都属于 H_0。

根据上述展开办法，H_0 首先应该包含 H_1 中的弧 a_3,a_{13},a_{14}。另外，应该把 U_1,U_2 展开成 C_1 和 C_2，并去掉 C_1 中与 H_1 中的 a_{13}有公共终点的 a_4 和去掉 C_2 中与 H_1 中的 a_8 有

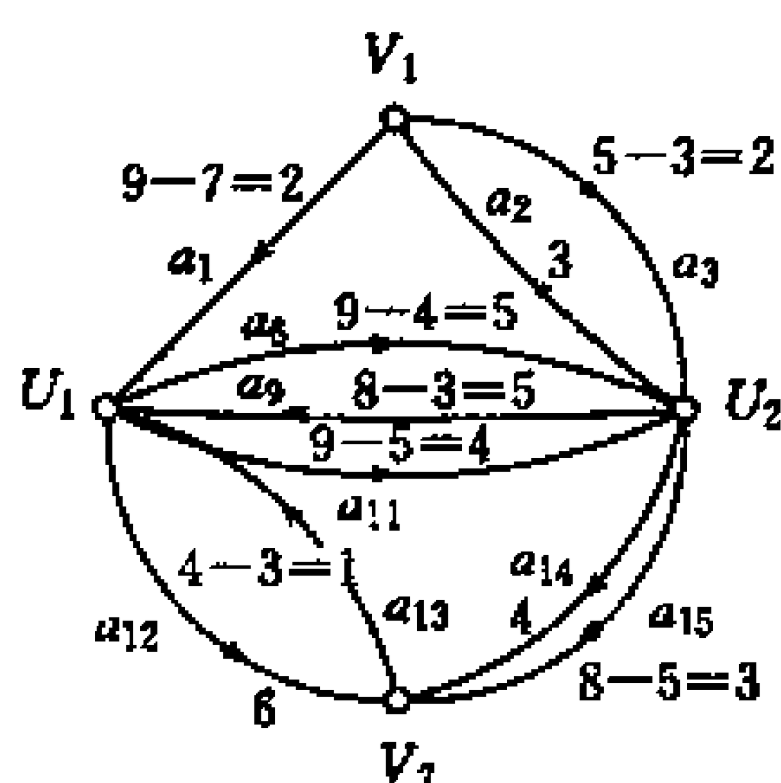


图 3-4

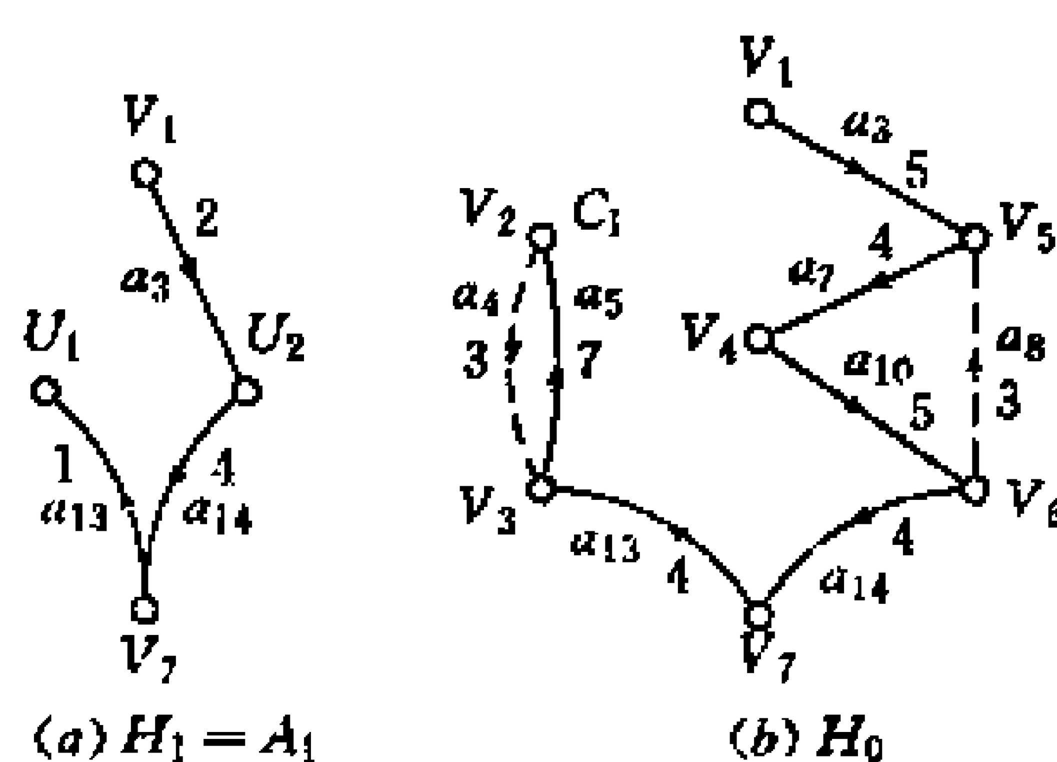


图 3-5

公共终点的 a_8，得到 H_0(见图 3-5(b)，图中虚线弧是展开时去掉的弧，它们不属于 H_0)。

求 D_0 的最小树形图 H_0 的算法可以简述成：

对 D_0 求最短弧集合 A_0。若 A_0 不存在，则 D_0 没有树形图。若 A_0 存在且不含有向圈，则 A_0 即为 D_0 的最小树形图 H_0；若 A_0 含有向圈，那么收缩 A_0 中的有向圈得到图 D_1。

对于 D_1 来说，如果求得的 A_1 还包含有向圈，那么还要收缩，得到 D_2；再把步骤 1，2，3 用到 D_2 上去，求出 A_2……。这样做下去，一直得到一个 D_t，它没有树形图(这时可以肯定 D_{t-1}，…，D_1，D_0 也都没有树形图)，或它对应的 A_t 不包含有向圈，即 A_t 是 D_t 的最小树形图 H_t。这时，通过步骤 4 的展开求出 D_{t-1}的最小树形图 H_{t-1}，再展开直至得到 D_0 的最小树形图 H_0 为止。

三、求最小树形图的程序

```
program shu_xing_tu;

  uses
    crt;

  const
    maxn          = 50;

  type
    graphtype     = array[1..maxn,1..maxn] of integer;
    ltype         = array[1..maxn] of integer;
    settype       = set of 1..maxn;
    setstype      = array[1..maxn] of settype;

  var
    g             : graphtype;     { 图的邻接矩阵 }
    cc            : setstype;      { 圈内的顶点序号 }
    n             : integer;       { 顶点数 }
    l,ft          : ltype;         { ft——D0 图的最小树形图 H0}
                                   { l——Di 顶点在 D0 中的序号 }
 f                : text;          { 文件变量 }
```

```
procedure init;
  var   i,j : integer;

        str : string;
    begin
      clrscr;
      write('file name = ');    { 读入文件名,并与文件变量连接 }
      readln(str);
      assign(f,str);
      reset(f);          { 读准备 }
      readln(f,n);     { 读入顶点数 }
      fillchar(g,sizeof(g),0); { 邻接矩阵初始化 }
      while not eof(f) do        { 读入各边的权 }
        readln(f,i,j,g[i,j]);
      close(f); { 关闭文件 }
    end;

procedure main;
  var i,j,k,t,x,y,v,u : integer;
      ft1 : ltype;             { D1 对应的树形图 H1 }
      more : boolean;          { 收缩标志 }
      ss : settype;            { H1 中的圈集合 }
    begin
      for i : =1 to n do l[i] : =i; { D1 各顶点在 D0 中的序号初始化 }
      fillchar(ft,sizeof(ft),0); { H0 初始化 }
      repeat more : =false;
             fillchar(ft1,sizeof(ft1),0); { H1 初始化 }
             for i : =1 to n do cc[i] : =[i];
            { 各顶点所在圈内的顶点集合初始化 }
             for v : =2 to n do
               if l[v]=v then { V 顶点目前未收缩 }
                 begin
                   k : =maxint;
                   for u : =2 to n do if l[u]=v then {若 u 顶点被收缩成 v}
                     for j : =1 to n do
                     { 在以 u 为终点的弧集中,取最短弧的起点 t }
                       if (l[j]<>v) and (g[j,u]>0) and (g[j,u]<k)
                       then begin k : =g[j,u]; t : =j; i : =u; end;
                   if k=maxint then { 若最短弧不存在,打印无解 }
                     begin writeln('no answer'); halt; end;
                   ft[i] : =t; ft1[v] : =l[t];
                   { t 顶点进入 H0,对应的 L[t]进入 H1}
                   if l[t] in cc[v] then
                   { 若 L[t]在以 v 顶点为收缩点所在的圈内 }
                     begin
                       more : =true;
                       k : =l[t]; { 圈内顶点的 D1 序号设为 v }
                       repeat l[k] : =v;
                              k : =ft1[k];
```

```
                    until l[k]=v;
                    for x:=1 to n do { 修正弧长 }
                      if l[x]<>v then
                        for y:=1 to n do
                          if (l[y]=v) and (g[x,y]>0) then
                            g[x,y]:=g[x,y]-g[ft[y],y];
                  end; { then }
                ss:=[];
                k:=v;
                repeat cc[ft1[k]]:=cc[ft1[k]]+cc[v];
                        { 设置圈内各顶点为收缩点的各圈集合 }
                        ss:=ss+[k];
                        k:=ft1[k];
            until (ft1[k]=0) or (k in ss);
          end; { then }
    until not more; { 直至不含有向圈为止 }
    for i:=2 to n do writeln(ft[i],'——',i); { 打印 H0}
  end;

begin
  init; { 输入有向图 }
  main; { 计算和输出最小树形图 }
end.
```

第四章　图的连通性

4.1　连通性的基本概念和定义

图的连通性这个概念十分重要，它的直观意义就是“连成一片”。当然，如果把连通图定义为“连成一片的图”就不太好，因为如果再追问一句：“什么叫连成一片？”恐怕就不好回答了。因此必须给图的连通性下一个准确的定义：

在无向图 G 中，如果从顶点 V 到顶点 V' 有路径，则称 V 和 V' 是连通的。如果对于图中任意两个顶点 $V_i, V_j \in V$，V_i 和 V_j 都是连通的，则称是连通图。

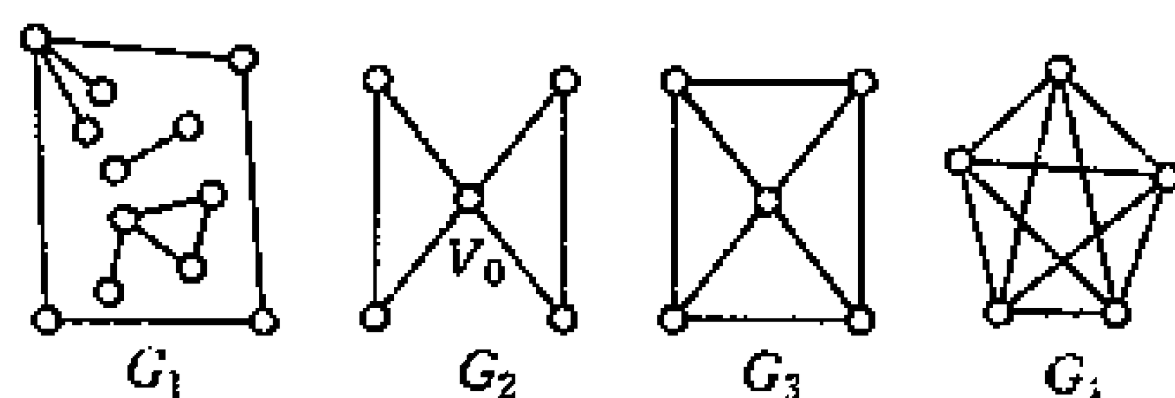


图　4-1

图 4-1 中的 G_2, G_3, G_4 是连通图，而 G_1 则是非连通图，但 G_1 有 3 个连通分量，如图 4-2 所示。所谓连通分量指的是无向图中的极大连通子图。

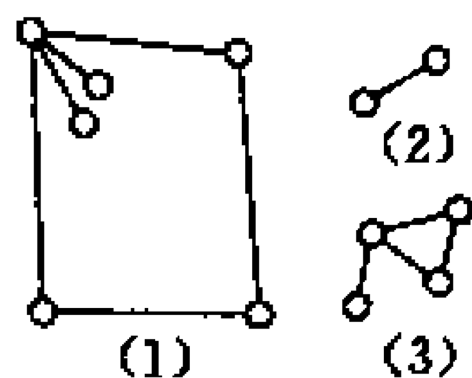


图　4-2

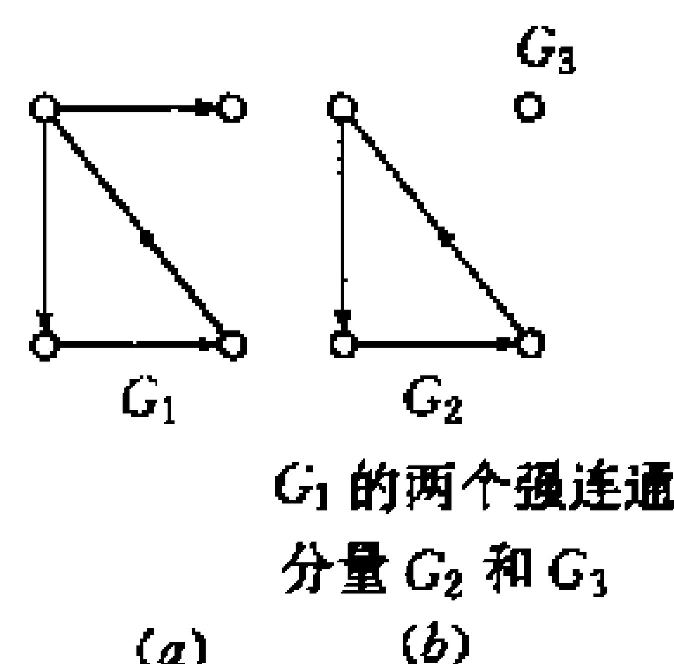


图　4-3

在有向图 G 中，如果对于每一对 $V_i, V_j \in V$，$V_i \neq V_j$，从 V_i 到 V_j 和从 V_j 到 V_i 都存在路径，则称 G 是强连通图。有向图中的极大强连通子图称作有向图的强连通分量。

例如图 4-3(a)中的 G_1 不是强连通图，但它有两个强连通分量，如图 4-3(b)所示。

一个连通图的生成树是一个极小连通子图，它含图的全部顶点，但只有足以构成一棵树的 $n-1$ 条边。图 4-4 是图 4-2 所示 G_1 中最大的连通分量的一棵树。如果在一棵生成树上添加一条边必构成一环，因为这条边使得它依附的那两个顶点之间有了第 2 条路径。一棵有 n 个顶点的生成树有且仅有 $n-1$ 条边。如果一个图有 n 个顶点而小于 $n-1$ 条边，则是非连通图，如果它多于 $n-1$ 条边，则一定有环。但是，有 $n-1$ 条边的图不一定是生成

树。

对于任意连通图来说，是否连通的程度都一样？不是。例如图 4-1 中的 G_2、G_3、G_4 都是连通图，但是连通的程度大不一样。G_2 删除一边后仍连通，但它有一个顶点 V_0，删去 V_0 和与其关联的各边后，G_2 就不连通了；G_3 只删除一条边或一个顶点后仍连通，而 G_4 是一个具有 5 个顶点的完全图，删除任意 3 个顶点或 3 条边都不能使其破碎。为精确刻画连通的程度，我们引入两个量：边连通度和顶连通度。

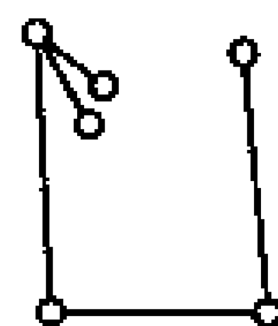


图4.2所示 G_1 的最大连通分量的一棵生成树

图 4-4

一、顶连通度 $K(G)$

V' 是连通图 G 的一个顶点子集。在 G 中删去 V' 及与 V' 相关联的边后图不连通，则称 V' 是 G 的割顶集。最小割顶集中顶点的个数，记成 $K(G)$，叫做 G 的连通度。规定 K(完全图)＝顶点数－1；K(不连通图)＝K(平凡图)＝0。$K(G)=1$ 时，割顶集中的那个顶点叫做割顶。没有割顶的图叫做块，G 中成块的极大子图叫做 G 的块。

例如图 4-1 中，$K(G_2)=1$，$K(G_3)=3$，$K(G_4)=5-1=4$；G_3 与 G_4 是块；G_2 中有两个三角形块。

二、边连通度 $K'(G)$

E' 是连通图 G 的一个边子集。在 G 中删去 E' 中的边后图不连通，则称 E' 是 G 的桥集。若 G 中已无桥集 E''，使得 $|E''|<|E'|$，则称 $|E'|$ 为 G 的边连通度，记成 $K'(G)$。$|E'|=1$ 时，E' 中的边叫做桥。规定 K'(不连通图)＝0，K'(完全图)＝顶点数－1。

例如图 4-1 中，$K'(G_2)=2$，$K'(G_3)=3$，$K'(G_4)=5-1=4$。G_2，G_3，G_4 中无桥。

对于任意一个连通图 G，在计算出上述两个量后，我们可以给该图下一个定义：

$K(G)\geqslant M$，G 叫做 M 连通图；$K'(G)\geqslant M$ 时，G 称为 M 边连通图。

例如图 4-1 中，G_2 是 1 连通图，2 边连通图，当然也是 1 边连通图；G_3 是 3 连通图，当然也是 1，2 连通图，G_3 是 3 边连通图，当然也是 1，2 边连通图；G_4 是 4 连通图，当然也是 1，2，3 连通图，G_4 是 4 边连通图，当然也是 1，2，3 边连通图。

M 连通图，当 $M>1$ 时，也是 $M-1$ 连通图。

M 边连通图，当 $M>1$ 时，也是 $M-1$ 边连通图。

下面给出连通图 G 的两个特征：

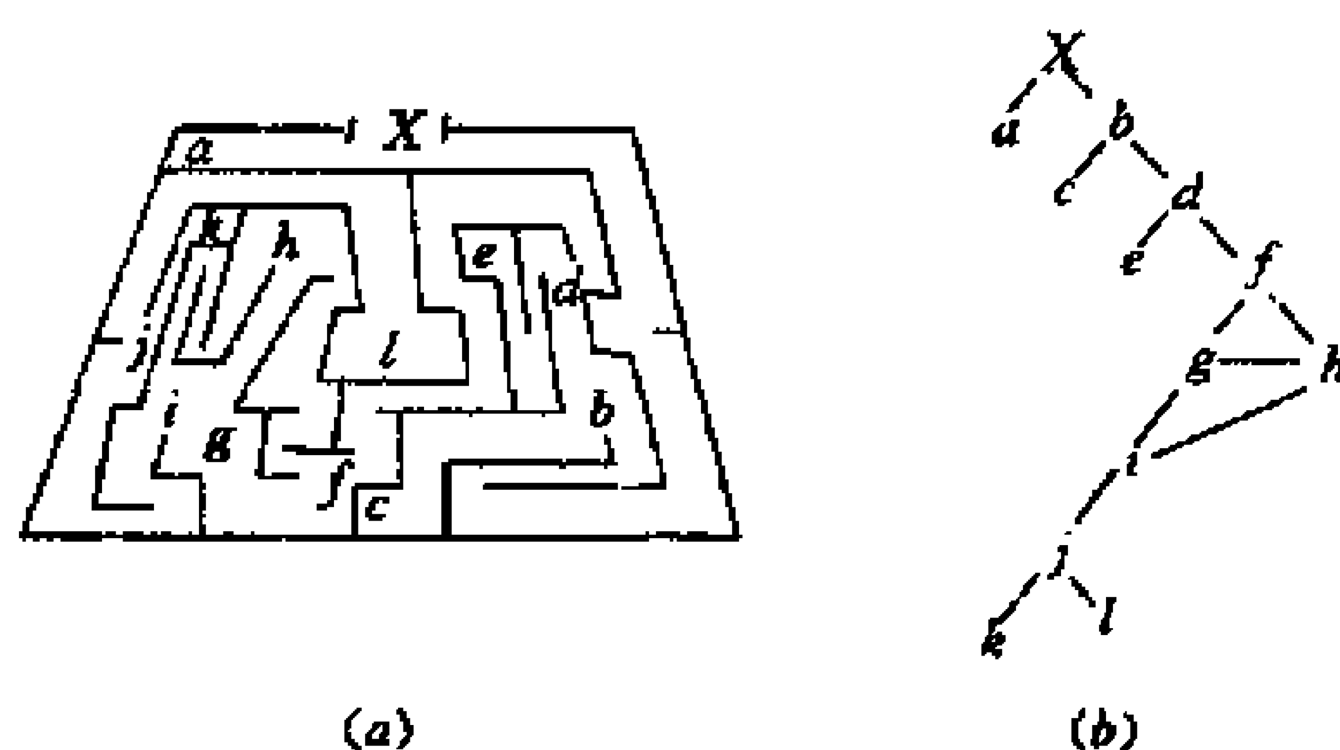
1. $K(G)\leqslant K'(G)\leqslant G$ 的顶点度数的最小值 &
2. 顶点数大于 2 的 2 连通图的充分必要条件是任两个顶点在一个圈上。

由于连通度和边连通图的计算涉及网络流知识，因此我们将在第六章作专门介绍。这一章里我们给出求割顶、块以及极大强连通子图的算法，并给出两个实际应用。在此之前，我们先介绍一个关键算法——深度优先搜索(dfs)。它在解图的连通性问题中扮演了极其重要的角色，而且其思想方法还渗透到其它许多图论算法的设计中。

4.2 深度优先搜索(dfs)

一、什么是深度优先搜索

为了理解深度优先搜索的思路，让我们追溯到1690年修筑的威廉王迷宫，它至今保存完好(见图4-5(*a*))。



(*a*) (*b*)

图 4-5

迷宫法则：任务是从迷宫入口处出发，每个走廊都要搜索，最后再从入口出来。为了不兜圈子，需要记住哪些走廊已经走过，沿着未走过的通道尽可能远地走下去，走到死胡同或那里已无未走过的走廊可选时，沿原路返回，到达一个路口，发现可通往一条未走过的走廊可选时，沿这一未走过的走廊尽可能远地走下去……，最后即可搜索遍全部走廊和厅室，再由入口处出迷宫。

我们把图4-5(*a*)中字母表示的厅室看作是顶点。如果两个厅室之间通有走廊，则在代表它们的两个顶点之间连一条边，两个厅室之间没有通道就不连。这样形成了一个图(图4-5(*b*))。

迷宫法则粗略地描述出深度优先搜索的算法轮廓。现在我们针对抽象出的无向图再作一番讨论。为了简略起见，先看连通图。对于非连通图，则只要在每个连通分量中用一次dfs算法便可遍历图的每个顶点。

先任取一个顶点作为根，记为U，现认为U顶点已访问过(在迷宫问题中，取入口X作为根)。再在U的邻接顶点中，任选W顶点，形成关联边(U,W)，且对这条边定方向为U到W。此时边(U,W)称为"已检查"，且送入树枝边集合中，U称为W的父亲点，记为$U=\text{father}(W)$。

一般情况下当访问某顶点X时，有两种可能：

1. 如果X顶点的所有关联边被检查过了，则回溯至X的父亲顶点，从father(X)再继续搜索，此时X顶点已无未用过的关联边；

2. 如存在X顶点未用过的关联边，则任选其中一边(X,Y)，对其定向为从X到Y，此时说边(X,Y)正等待检查。

检查结果有两种情况：

1. 如Y顶点从未被访问过，则顺着(X,Y)边访问Y，下一步从Y开始继续搜索。此时

(X,Y)是前向边，且顶点 $X=\text{father}(Y)$，图上用实线表出；

2. 如 Y 顶点已被访问过，则再选 X 顶点未用过的关联边，此时(X,Y)是后向边，用虚线表出。

此时，(X,Y)的归属已明确，(X,Y)就检查完毕。在 dfs 搜索期间，根据顶点 X 被首次访问的次序安排了不同的整数 dfn(X)，如 dfn$(X)=i$，则 X 是第 i 个首次被访问的顶点，dfn(X)称为 X 的深度优先搜索序数，当搜索返回根时，连通图的所有顶点都访问完毕。现在图中的边都定了方向，且分成前向边(实线)组成的有向树以及后向边(虚线)，这个有向生成树称为 dfs 树。

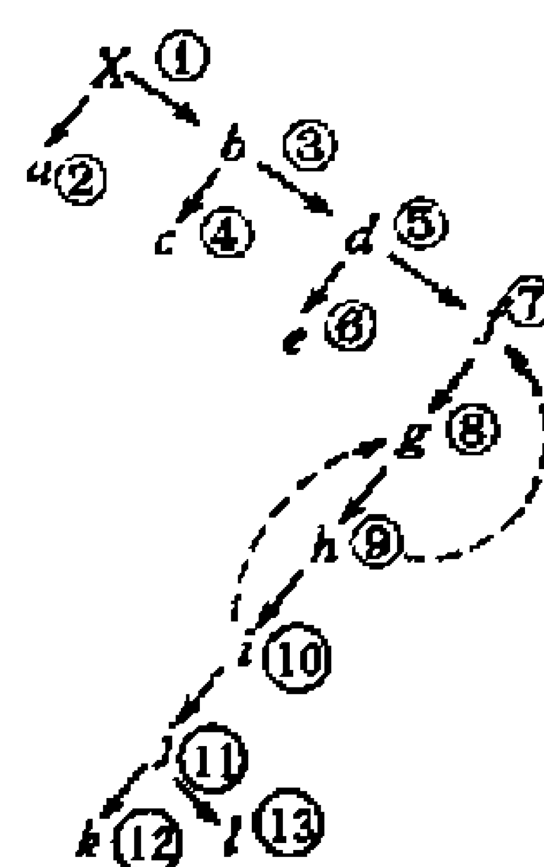
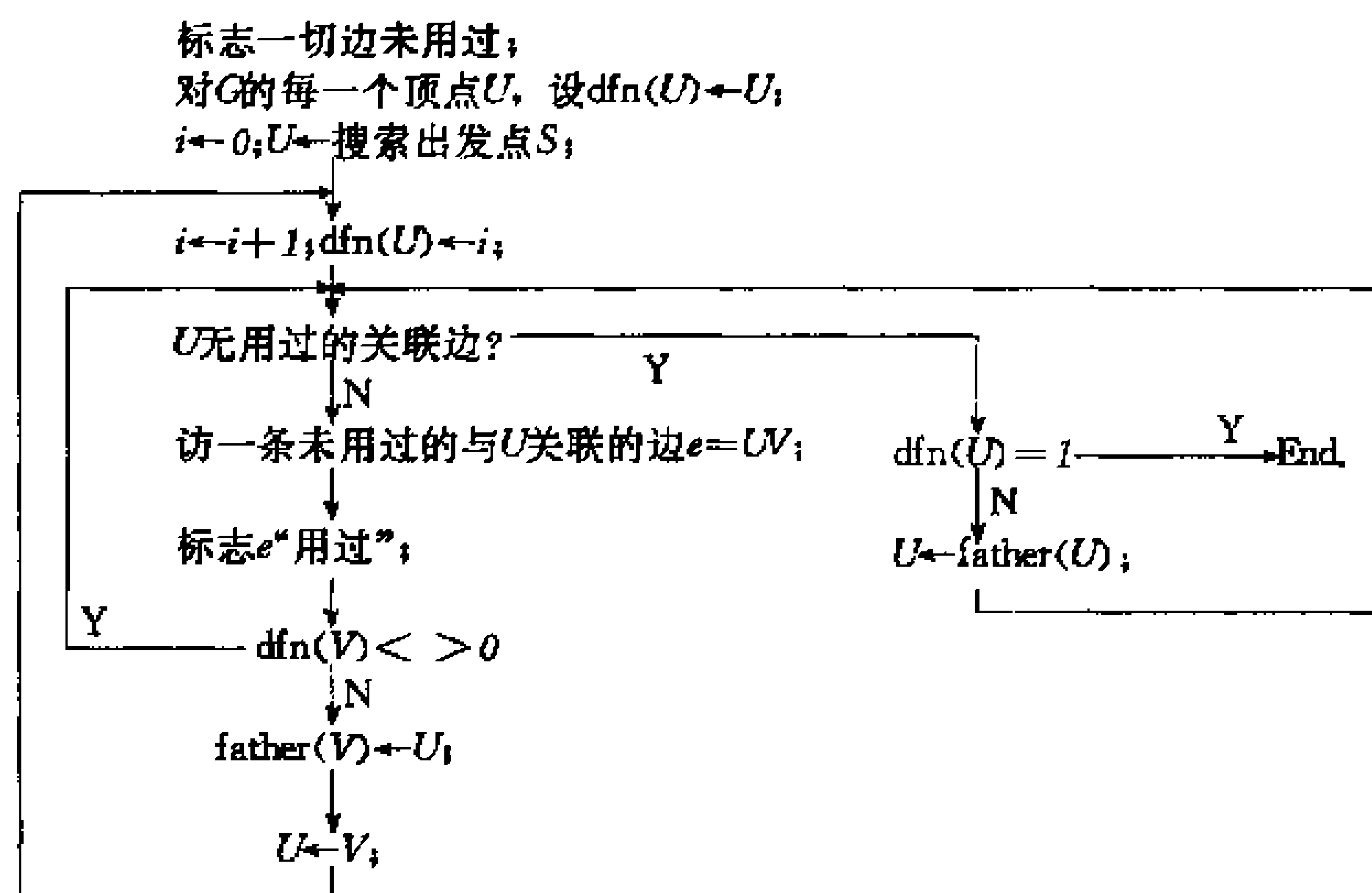


图 4-6

图 4-6 给出图 4-5(b)的 dfs 搜索过程。每个顶点旁○内的数字为 dfn 值。

显然 dfs 算法类似树的前序遍历。

二、无向图 G 的 dfs 算法分析



为了便于在以后各章节中深入开发上述算法，再引进一些术语。现在 G 从无向图改造成有向图，且每顶点有 dfn 值，前向边组成 dfs 树，(若非连通图，则对每一个连通分量进行一次 dfs 搜索，结果产生一个由若干 dfs 树组成的森林)。如果从 U 到 W 有树边组成的有向路，则称 U 和 W 有直系关系，U 称为 W 的祖先点，W 称为 U 的后代点。如(U,W)是树中一条边，U 又确切地称为 W 的父亲点，W 也可称为 U 的儿子点。一个顶点可有多个儿子点，顶点 U 和它的所有后代点组成关于树中以 U 为根的子树。

如果 U 和 W 不存在直系上下关系，则说它们有旁系关系。在这种情况下，dfn$(U)<$ dfn(W)，则说 U 在 W 的左边，或 W 在 U 的右边。连接旁系之间的边叫横叉边。在 dfs 搜索中，G 不产生横叉边。或者说：如(U,W)是无向图 G 的一条边，则任何一种 dfs 树(取不同的根可有不同的树)，U 或是 W 的祖先点，或是 W 的后代点。

为什么？请读者自行分析。

下面讨论有向图的 dfs。

有向图的 dfs 本质上近似无向图，区别在于搜索只能顺边的方向去进行(有向边，简称弧)。有向图中的弧根据被检查情况可有 4 种，一条未检查的弧(U,W)可按如下 4 种情况，分类归划成(其中 2.，3.，4. 所述的三种情况中，设 W 已访问过)：

1. W 是未访问过的点，(U,W)归入树枝弧(见图 4-7(a))

2. W 是 U 的已形成的 dfs 森林中直系后代点，则(U,W)称为前向弧(见图 4-7(b))。无论(U,W)是树枝弧还是前向弧，$\mathrm{dfn}(W)>\mathrm{dfn}(U)$。细分一下，树枝边总使搜索导向一个新的(未访问)点，且 $\mathrm{dfn}(U)+1=\mathrm{dfn}(W)$。

3. W 是 U 的已形成的 dfs 森林中直系祖先点，则(U,W)称为后向弧(见图 4-7(c))。

4. U 和 W 在已形成的 dfs 森林中没有直系上下关系，并且有 $\mathrm{dfn}(W)<\mathrm{dfn}(U)$，则称$(U,W)$是横叉弧。注意没有 $\mathrm{dfn}(W)>\mathrm{dfn}(U)$ 这种类型的横叉弧。因为 dfs 搜索只能从左到右逐条树枝地进行(见图 4-7(d))。

无论(U,W)是后向弧(W 的所有关联边未完全检查)还是横叉弧(W 的所有关联边完全检查)，$\mathrm{dfn}(W)>\mathrm{dfn}(U)$。



图 4-7

例如图 4-8 所示的有向图经 dfs 后，弧划分为：

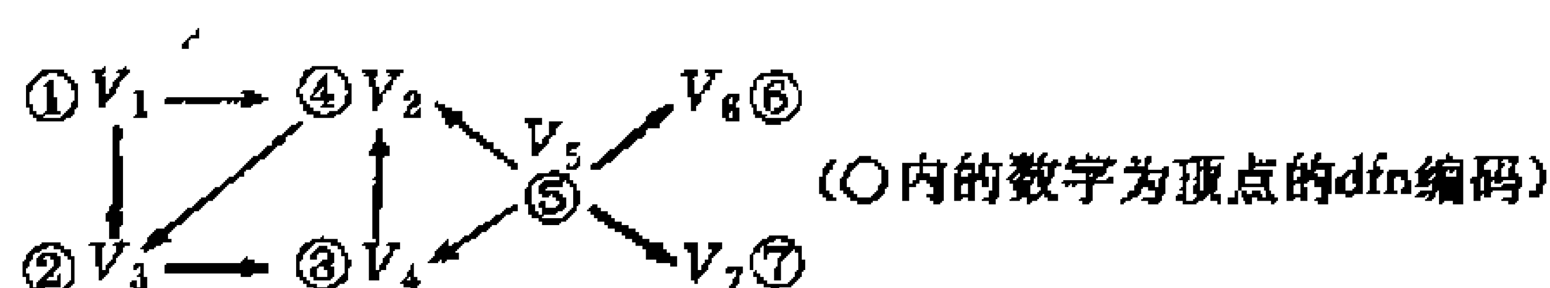
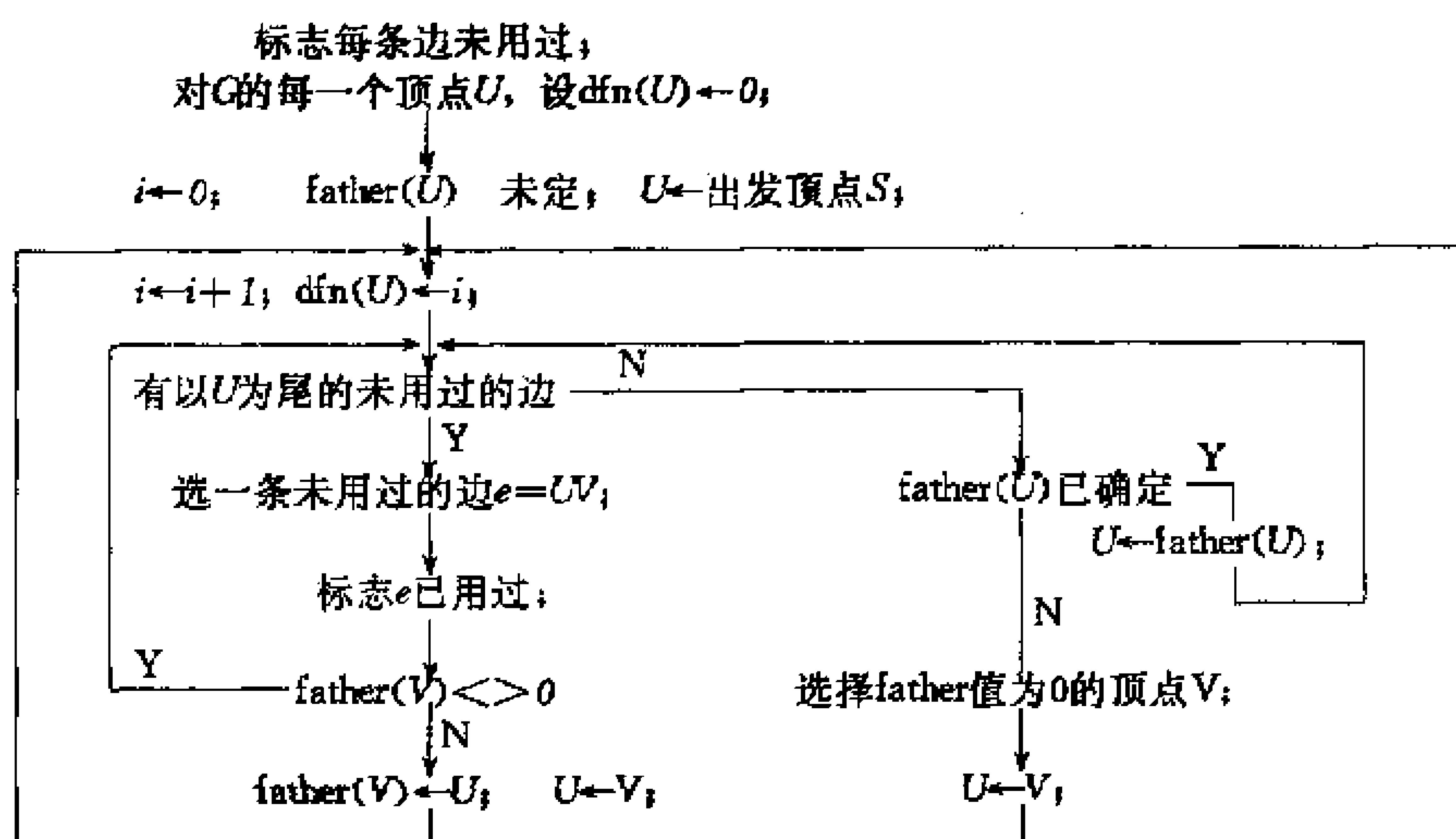


图 4-8

$<V_1V_3>$，$<V_3V_4>$，$<V_4V_2>$，$<V_5V_6>$，$<V_5V_7>$是树枝弧，组成 dfs 森林(用粗线表示)；$<V_2V_3>$是后向弧；$<V_1V_2>$是前向弧；$<V_5V_2>$，$<V_5V_4>$是横叉弧。

下面给出有向图 G 的 dfs 算法：



三、无向图的 dfs 程序

由于有向图的 dfs 本质上近似无向图，况且这个算法将在 4.4 节求极大强连通子图中还会使用到，因此这里不再作介绍。

```
program dfs_p94;

  const
    maxn        =  30;

  type
    node        =  record
                     k,f  :  1..maxn { 顶点类型： DFS 编码，父指针 }
                   end;
    ghtype      =  array [1..maxn,1..maxn] of integer;
    ltype       =  array [1..maxn] of node;

  var
    g          :  ghtype;      { 邻接矩阵 }
    n          :  integer;     { 顶点数 }
    f          :  text;        { 文件变量 }
    l          :  ltype;       { 顶点序列 }

  procedure read_graph;
    var
      str : string;
      i,j : integer;
    begin
      write('Graph file = ');   { 读入外部文件名并与文件变量连接 }
      readln(str);
      assign(f,str);
      reset(f);
      readln(f,n);  { 读入结点数 }
```

```
    for i := 1 to n do { 读入图矩阵 }
      for j := 1 to n do read(f,g[i,j]);
    close(f); { 关闭文件 }
  end;

procedure DFS(s : integer);
  var
    i,v,u : integer;
  begin
    fillchar(l,sizeof(l),0); { l 数组清零 }
    i := 0;v := s; { s 作为第一个访问结点，即深度优先树的根 }
    repeat
      repeat
        inc(i);l[v].k := i; { 设置 v 顶点的 dfn 编码 }
          repeat
            u := 1;
            while (u<=n) and (g[v,u]<=0) do inc(u);
            { 选一条未用过的与 v 关联的边<v,u> }
            g[v,u] := -g[v,u];g[u,v] := -g[u,v]; { 标志<u,v>用过 }
          until (u=n+1) or (l[u].k=0);
          { 直至与 v 关联的所有边都使用过或者<v,u>的顶点 u 未曾编码为止 }
          if u<>n+1                        { 若<v,u>的顶点 u 未曾编码,则 }
            then begin
                   writeln(v,'——',u);
                     { 打印边<v,u>,设置 u 的父亲结点为 v,从 u 出发搜索 }
                   l[u].f := v;v := u
                 end
        until u=n+1; { 直至与 v 关联的边都使用过 }
      if l[v].k<>1 then v := l[v].f { v 不是根结点,回溯至 v 的父亲结点 }
    until l[v].k=1 { 直至回溯至根结点 }
  end;
begin
  read_graph; { 输入图 }
  DFS(1)       { 从顶点 1 出发,作深度优先搜索 }
end.
```

4.3 求割顶和块

一、什么是割顶和块

分析一个由无向图表示的通讯网,顶点看作是通讯站,无向边代表两个通讯站之间可以互相通讯。问:

1. 若其中一个站点出现故障,是否会影响系统正常工作;

2. 这个通讯网可以分成哪几个含站点尽可能多的子网,这些子网内的任一站点出现故障,都不会影响子网工作。

有了连通性的基础知识后,我们不难得出,问题 1 所指的那个“牵一发动全身”的站点,称为割点。在删去这个顶点以及和它相连的各边后,会将图的一个连通分割成两个或

两个以上的连通分量，影响系统正常工作；而问题2所指的那些内部任一站点出现故障都不会影响其通讯的最大子网，指的是没有割点的连通子图。这个子图中的任何一对顶点之间至少存在两条不相交的路径，或者说要使这个子网不连通，至少要两个站点同时发生故障。这种二连通分支也称为块。

显然各个块之间的关系或者互不连接，或者通过割顶连接。这割顶可以属于不同的块，也可以两个块共有一个割顶。因此无向图寻找块，关键是找出割顶。下面有两个基本事实：

对一个给定的无向图，实施dfs搜索得到所有顶点的dfn值及dfs树（或森林）后，

1. 如U不是根，U成为割顶当且仅当存在U的某一个儿子顶点S，从S或S的后代点到U的祖先点之间不存在后向边（见图4-9(a)）；

2. 如γ被选为根，则γ成为割顶当且仅当它有不止一个儿子点（见图4-9(b)）。

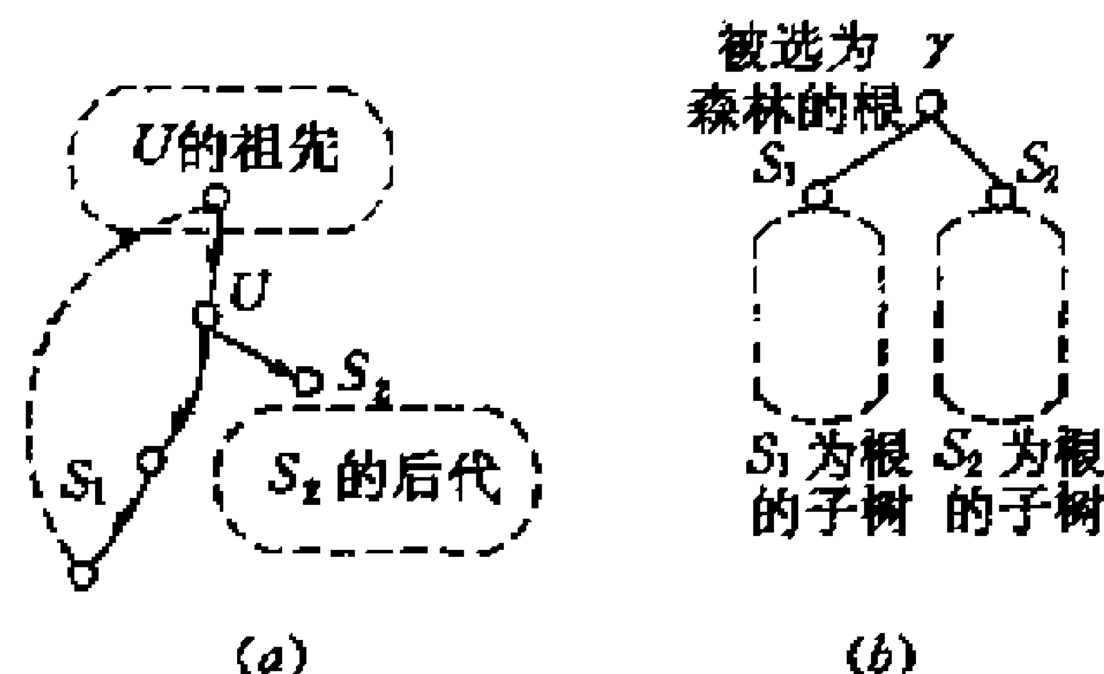


图 4-9

由图4-9(a)可以看出，虽然U的以S_1为根的子树中有后向边返回U的祖先，但是U的某儿子S_2或S_2的后代没有返回U的祖先，因此由基本事实1得出U是割点。

由图4-9(b)可以看出，根γ存在两棵分别以S_1和S_2为根的子树，这两棵子树间没有横叉边（无向图不存在横叉边），因此由基本事实2得出γ是割顶。

上述两个基本事实是构思算法的精华，为此引入一种顶点U的标号函数LOW(U)

$$\text{LOW}(U)=\min(dfn(U),\text{LOW}(S),dfn(W))$$

其中：S是U的一个儿子，(U,W)是后向边。

显然LOW(U)值恰是U或U的后代所能追溯到的最早（序号小）的祖先点序号。一般约定，顶点自身也认为自己是祖先点。所以有可能$\text{LOW}(U)=dfn(U)$或$\text{LOW}(U)=dfn(W)$。

利用标号函数LOW，我们可以将基本事实1重新描述成：

顶点$U\neq\gamma$作为G的割顶当且仅当U有一个儿子S，使得$\text{LOW}(S)\geqslant dfn(U)$，即$S$和$S$的后代不会追溯到比$U$更早的祖先点。

LOW(U)值的计算步骤如下：

$$\text{LOW}(U)=\begin{cases}\text{—dfn}(U) \\ \quad U\text{ 在 dfs 过程中首次被访问；} \\ \text{—min}(\text{LOW}(U),\text{dfn}(W)) \\ \quad\text{检查后向边}(U,W)\text{时；} \\ \text{—min}(\text{LOW}(U),\text{LOW}(S)) \\ \quad U\text{ 的儿子 }S\text{ 的关联边全部被检查时。}\end{cases}$$

在算法执行中，对任何顶点U计算LOW(U)值是不断修改的，只有当以U为根的dfs子树和后代的LOW值、dfn值产生后才停止。

二、求割顶和块的算法

根据前面的分析求割顶和块的算法如下面的逻辑图所表示。

标志一切边未用过；stack栈置空；所有顶点的dfn值置0；
$i \leftarrow 0$；$U \leftarrow$ 初始顶点S；

$i \leftarrow i+1$；$dfn(U) \leftarrow i$；$Low(U) \leftarrow i$；U入栈stack；

U有未用过的关联边？（N → dfn(father(U))=1；Y ↓）

选一条未用过的边$e=UV$；标志e用过；

$dfn(V) <> 0$ —— Y（UV是后向边）→ $Low(U) \leftarrow \min\{Low(U), dfs(V)\}$

（UV是前向边）N ↓

$father(V) \leftarrow U$；$U \leftarrow V$；

$dfn(father(U))=1$ —— Y → 将stack栈中连同U在内U上方的顶点移出……

N ↓

$Low(U) < dfn(father(U))$ —— Y → $Low(father(U)) \leftarrow \min\{Low(father(U)), Low(U)\}$；

N ↓

由基本事实1得出father(U)是割顶。
将stack栈中包括U在内的U上部的顶点移出，这些顶点连同father(U)导出一个块；

$U \leftarrow father(U)$；

将stack栈中连同U在内U上方的顶点移出，
这些顶点连同初始顶点S导出一个块；

初始顶点S无未用过的关联边 —— Y → end.

N ↓

由基本事实2得出初始顶点S是割顶；

$U \leftarrow$ 初始顶点S；

例如图 4-10(a)

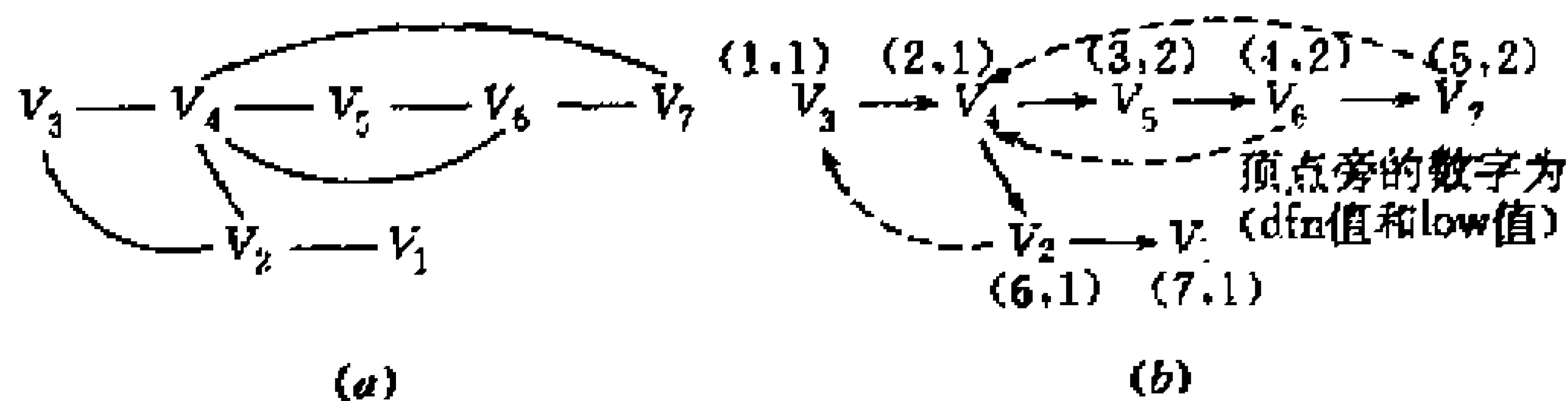


图 4-10

从 V_3 出发，按上述算法进行搜索，得出 dfs 树和各顶点 dfn 值、LOW 值（见图 4-10(b)）。由于$\langle V_4V_5 \rangle$是前向边，且 $dfn(V_4)=2>1$，$LOW(V_5)=2$，又 $LOW(V_4)=2$，由基本事实 1 得出 V_4 是割顶，并导出$\{V_4, V_3, V_2\}$和$\{V_4, V_5, V_6, V_7\}$两个块。

三、求割顶和块的程序

```
Program QIU_KUI_HE_GE_DING;

const
    maxn          =  30;

type
  node            =  record
                       k,i : 1..maxn { 顶点类型: dfn 编码,low 编码 }
                     end;
  ghtype          =  array [1..maxn,1..maxn] of integer; { 邻接矩阵类型 }
  ltype           =  array [1..maxn] of node;

var
  g               :  ghtype; { 邻接矩阵 }
  n,p,x,i         :  integer; { 顶点数,栈指针,根的子树个数,辅助变量 }
  f               :  text;    { 文件变量 }
  l               :  ltype;   { 顶点序列 }
  st              :  array [1..maxn] of integer; { 栈 }
  k               :  set of 1..maxn; { 割顶集 }

function min(a,b : integer) : integer; { 返回 a,b 间的小者 }
  begin
    if a>=b then min :=b
             else min :=a
  end;

procedure push(a : integer); { a 入栈 st }
  begin
    inc(p);st[p] :=a
  end;

function pop : integer; { 返回 st 的栈顶元素,退栈 }
  begin
    pop :=st[p];dec(p)
  end;

procedure read_graph;
  var
    str : string;
    i,j : integer;
  begin
    write('Graph file = '); { 输入外部文件名并与文件变量连接 }
    readln(str);
    assign(f,str);
    reset(f);
    readln(f,n); { 读入顶点数 }
    for i :=1 to n do  { 读入图矩阵 }
```

```
      for j:=1 to n do read(f,g[i,j]);
    close(f);
    fillchar(l,sizeof(l),0); { l 数组初始化 }
    p:=0;push(1);              { 顶点 1 入栈 }
    i:=1;l[1].k:=i;l[1].l:=i; { 顶点 1 首次被访问 dfs(1)=low(1)=1 }
    k:=[];x:=0 { 割顶集置空 }
  end;

procedure algorithm(v:integer);
  var
    u:integer;
  begin
    for u:=1 to n do
      if g[v,u]>0 { 若<v,u>边存在,则分析 }
        then if l[u].k=0 { 若 u 首次被访问 }
              then begin
                    inc(i);l[u].k:=i;l[u].l:=i;push(u);
                    { 置 u 的 dfs 和 low 值; u 入栈 }
                    algorithm(u);
                    { 从 u 结点出发,递归求解 }
                    l[v].l:=min(l[v].l,l[u].l);
                    { v 的儿子 u 完全扫描毕,置 v 的 low 值 }
                    if l[u].l>=l[v].k
                      { 从 u 及 u 的后代不会追溯到比 v 更早的祖先点 }
                      then begin
                            if (v<>1) and not (v in k)
                   { 若 v 不是根且不属割顶集,则打印割顶 v,v 进入割顶集 }
                              then begin
                                    writeln('[',v,']');
                                    k:=k+[v]
                                  end;
                            if v=1 then inc(x);
                   { 若 v 是根,计算 v 根下的子树个数 }
                            while st[p]<>v do write(pop:3);
                   { 从栈中依次取出至 v 的各顶点,产生一个连通分支 }
                            writeln(v:3)
                          end
                   end
              else l[v].l:=min(l[v].l,l[u].k)
                   { <v,u>是后向边,u 重复访问,则置 v 的 low 值 }
  end;

begin
  read_graph;  { 输入图 }
  algorithm(1); { 递归求解割顶集与块 }
  if x>1 then write('[',1,']'); { 若根不止有一个儿子,则根作为割顶打印 }
end.
```

4.4　求极大强连通子图

一、什么是极大强连通子图

让我们先看一个例子。图 4-11 画了一个有向图，它的每一个顶点代表某篮球队的一个队员，它的弧代表的意思是如果有一条从 V_i 出发而指向 V_j 的弧(i,j)，就表示队员 V_i 能够通知 V_j。

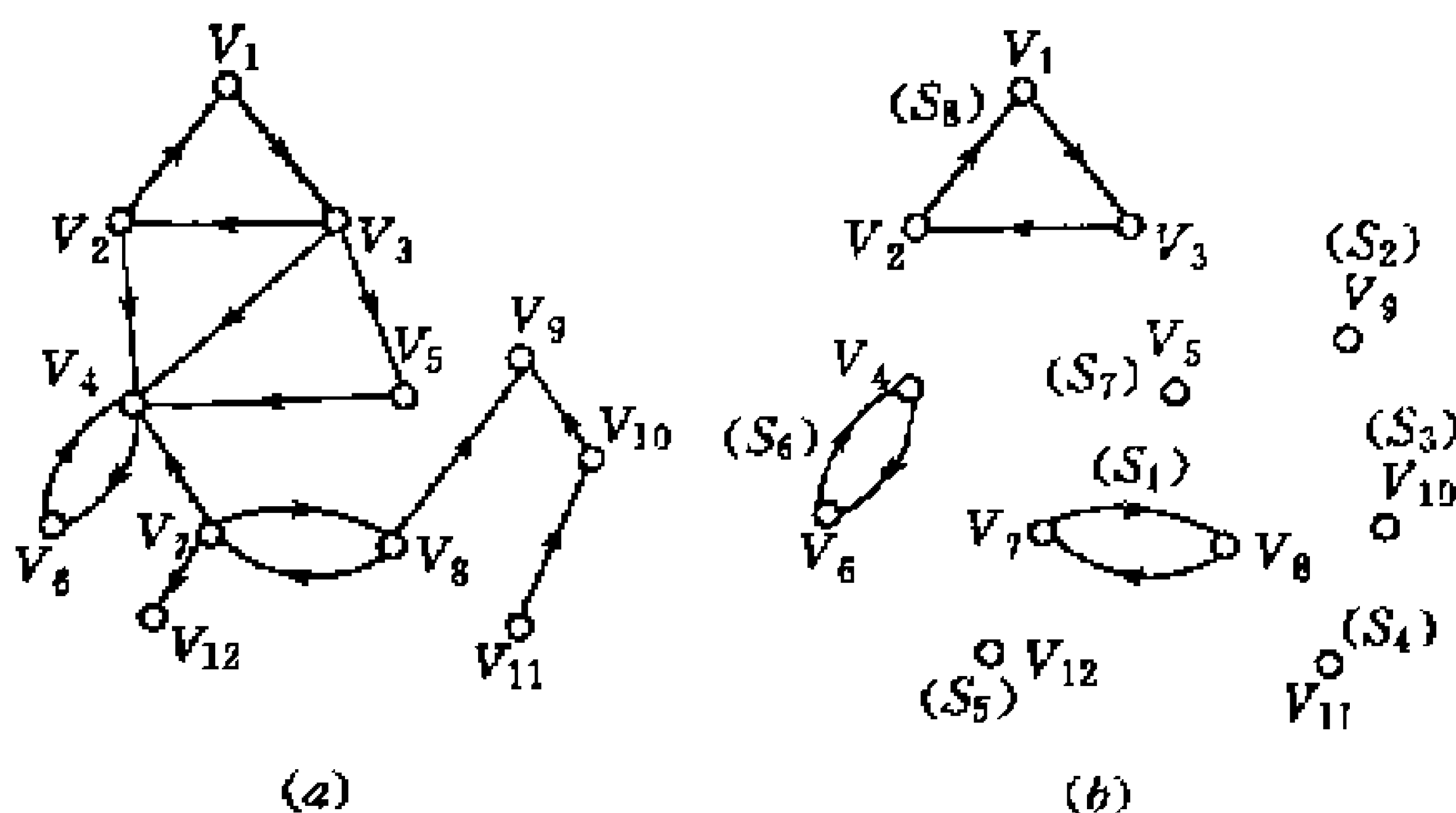


图　4-11

问这张有向图至少能划分几个最大子图，这些子图上的所有队员能相互通知（包括子图仅含一个队员，表示自己通知自己）。

有了连通性的基本知识后，我们不难得出，上述问题要求的最大子图即为极大强连通子图，在这个子图上的任一对顶点可以互相可达。按这个要求划分子图，图 4-11(a)的有向图共有 8 个强连通分支，它们分别是由

$$S_1=\{V_7,V_8\} \qquad S_2=\{V_9\}$$
$$S_3=\{V_{10}\} \qquad S_4=\{V_{11}\}$$
$$S_5=\{V_{12}\} \qquad S_6=\{V_4,V_6\}$$
$$S_7=\{V_5\} \qquad S_8=\{V_1,V_2,V_3\}$$

生成的（见图 4-11(b)）。

那么，如何在有向图中寻找任意两个顶点都可用有向路双向连通的最大子图呢？还是采用 dfs 算法。因为每一个强连通子图从 dfs 树看，都是以 dfn 值最小的顶点为根的子树，这个根亦被称作强分支的根。

为了在 dfs 森林的基础上找强分支子树的根，我们引入了顶点 U 的标值函数 LOWLINK(U)：

$$\text{LOWLINK}(U)=\min\{\text{dfn}(U),\text{dfn}(W)\}$$

其中 W 是从 U 或 U 的后代点出发用一条后向弧或横叉弧所能到达的同一强连通分支中的顶点。

由此可以看出，LOWLINK(U)正是 U 所处的强分支中从 U 出发先用树枝弧、前向弧，最后用后向弧或横叉弧到达的 dfn 最小的顶点的序值。而对于强分支的根 r_i 显然有

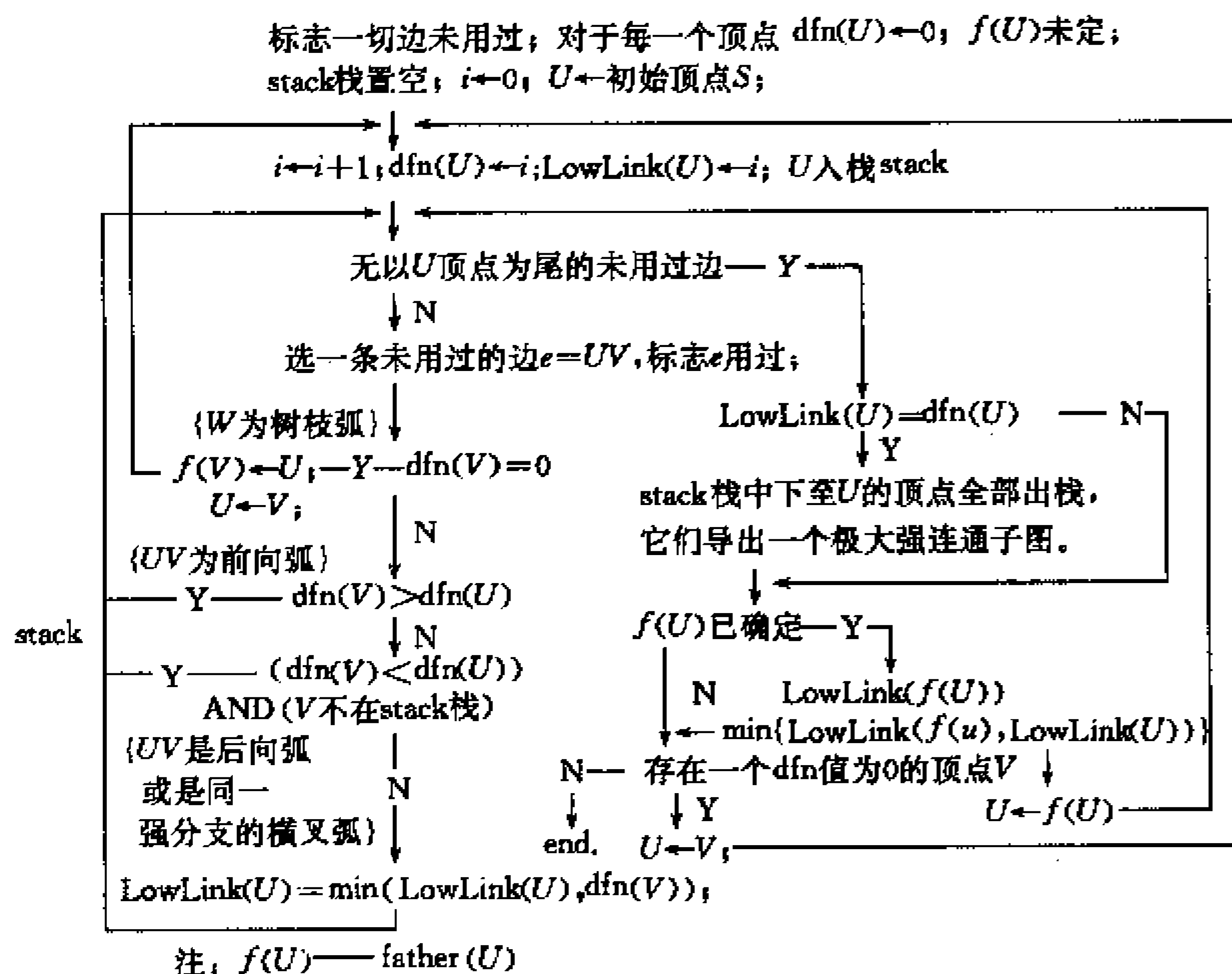
LOWLINK(r_i)=dfn(r_i)。因此当深度优先搜索从一个使 dfn 编码=LOWLINK 值的顶点 U 返回时，从树中移去根为 U 的所有顶点。每一个这种顶点的集合是一个强分支，即极大强连通子图。

算法的主要思路是用逐步迭代计算出 LOWLINK(U)值。

LOWLINK(U)=
- —dfn(U)
  第一次访问顶点 U 时；
- —min(LOWLINK(U),dfn(W))
  后向弧(U,W)被检查时；
- —min(LOWLINK(U),dfn(W))
  处于同一强分支的横叉弧(U,W)被检查时；
- —min(LOWLINK(U),LOWLINK(S))
  检查了 U 的儿子 S 的所有关联边后返回顶点 U 时；

二、求极大强连通分支的算法

下面介绍求极大强连通分支的算法。



注：f(U)—— father(U)

三、求极大强连通分支的程序

```
program qiu_ji_da_qiang_liang_tong_de_suan_fa;


  const
    maxn          =   30;
```

```
type
  node        =  record
                   k,l : 1..maxn { 顶点的 dfs 和 lowlink 值 }
                 end;
  ghtype      =  array [1..maxn,1..maxn] of integer;
  ltype       =  array [1..maxn] of node;

var
  g           :  ghtype;  { 邻接矩阵 }
  n,p,i,x     :  integer; { 顶点数,栈指针,辅助变量,子树个数 }
  f           :  text;     { 文件 }
  l           :  ltype;   { 顶点序列 }
  st          :  array [1..maxn] of integer;
                 {存储当前强连通分支顶点的栈 }

procedure read_graph;
  var
    str : string;
    i,j : integer;
  begin
    write('Graph file = '); { 读入外部文件名,并与文件变量连接 }
    readln(str);
    assign(f,str);
    reset(f);
    readln(f,n); { 读入顶点数 }
    for i :=1 to n do
      for j :=1 to n do read(f,g[i,j]); { 读入邻接矩阵 }
    close(f);
    fillchar(l,sizeof(l),0); { 顶点信息初始化 }
    p :=1;
  end;

function min(a,b : integer) : integer; { 返回 a,b 间的小者 }
  begin
    if a>=b then min :=b
            else min :=a
  end;

function check(a : integer) : boolean;
{ st 栈中存在 a 返回 false,否则返回 true }
  var
    i : integer;
```

```
  begin
    check :=true;
    for i :=1 to p do if a=st[i] then exit;
    check :=false
  end;

procedure require(v : integer); { 求以 v 为根的强连通分支 }
  var
    u : integer;
  begin
    inc(i);l[v].k :=i;l[v].l :=i;   {置 v 结点的 lowlink 和 dfs 值}
    inc(p);st[p] :=v;               {v 结点入栈}
    for u :=1 to n do
      if g[v,u]>0                   { 若<v,u>存在 }
         then if l[u].k=0           { 若 u 未检查,即<v,u>是前向弧 }
                then begin
                       require(u); { 从 u 出发,递归求解 }
                       l[v].l :=min(l[v].l,l[u].l);
                       { v 的儿子 u 被检查完毕,置 v 的 lowlink 值 }
                     end
                else if (l[u].k<l[v].k) and check(u)
{ <v,u>是后向弧或横向弧、u 被检查过且 u 与 v 处于同一强分支中,则修改 v 的 lowlink 值 }
                          then l[v].l :=min(l[v].l,l[u].k);
     if l[v].l=l[v].k { 若关联于 u 的全部弧都检查且 v 是强连通分支中的根 }
        then begin
                 repeat { st 栈中至 v 的顶点相继出栈,组成一个强连通分支 }
                   write(st[p] : 3);dec(p)
                 until st[p+1]=v;
                 writeln
               end
   end;

  procedure proceed;
    var
      x : integer;
    begin
      i :=0;p :=0;
      for x :=1 to n do if l[x].k=0 then require(x)
      { 对每一个未访问的顶点,求以它为树根的强连通分支 }
    end;

  begin
```

```
  read_graph; { 输入图 }
  proceed     { 计算输出所有强连通分支 }
end.
```

4.5 求最小点基

一、基本概念

还是把 4.4 节求极大强连通子图中的图 4-11(a)看作是某篮球队的通讯图。现在,我们对这张图提两个问题:

1. 教练想要通知全体队员都来练球,请你帮教练考虑一下,他至少要通知几个队员(然后由这些队员转告其他队员),才能使所有队员都被通知到;

2. 教练通知队员 U_i 时必须付出一定的代价 A_i(例如 A_i 可以代表教练给 V_i 打电话所需的时间),请你再帮教练考虑一下,如何以最小的代价使所有队员都得到通知。

图 4-12 给出了一种方案:教练直接通知 V_1,V_6,V_7,V_{11},然后由他们转告其他队员,全部队员就都能接到通知了。但是,是不是至少通知 4 个队员呢?能不能再少通知几个呢?

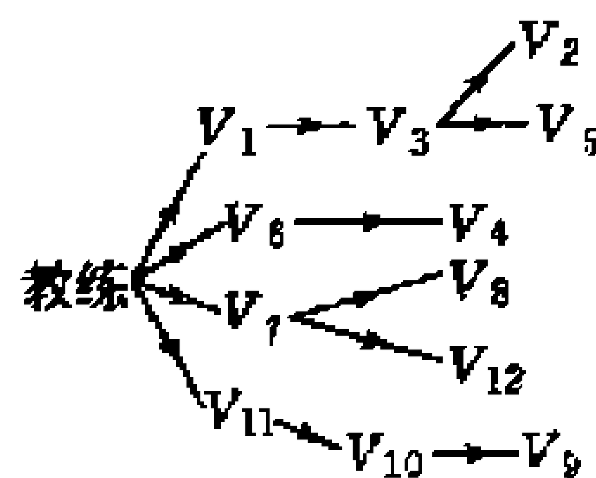


图 4-12

大家试一下就会知道,通知 3 个人就够了,例如可以只通知 V_1,V_7 和 V_{11},而再少就不可能了(想想看,为什么)。

上面这个例子很简单,通过简单的分析、试验就可以知道至少要通知三个人。但是如果遇到一个类似的问题,而图中的顶点很多,仅仅靠试验就不行了。必须要有系统的方法。这一节的主要目的就是介绍解决这类问题的一种方法。

首先,让我们把要解决的问题明确一下。在有向图中,如果存在一条从顶点 V_i 到 V_j 的有向路,就称 V_i 是 V_j 的前代,V_j 是 V_i 的后代。我们从这个概念出发考虑问题。因为如果教练通知了 V_i,那么所有 V_i 的后代 V_j 就都可以间接得到通知了,而要保证所有队员都得到通知,由教练直接通知的队员集合 B 必须具备下述性质:"对于任意一个队员 V_j,一定存在 B 中的一个 V_i,使得 V_i 是 V_j 的前代"。或者说,B 的所有后代包括了 G 的所有顶点。这个 B 集合就是所谓的点基:

设 $G=(V,E)$ 是一个有向图,B 是若干个顶点组成的 V 的子集。如果对于任意的 $V_j\in V$,都存在一个 $V_i\in B$,使得 V_i 是 V_j 的前代,则称 B 是一个点基。

例如按刚才讲的两个方案 $B_1=\{V_1,V_6,V_7,V_{11}\}$ 及 $B_2=\{V_1,V_7,V_{11}\}$ 都是点基。

前面讲的两个问题,实际上是求两个特殊要求的点基。问题 1 可以归结为求一个包含顶点数最少的点基(即最小点基),而问题 2 又可以归结为另一类点基:

设图 G 的每一顶点 V_i 都对应一个非负数 A_i(即 V_i 的权),现在要求一个点基,使得它所包含的顶点对应的 A_i 之和最小(即权最小的点基)。

注意,如果令每一个顶点 V_i 的权 A_i 都等于 1,那么权最小的点基就是最小点基。因此权最小点基是最小点基的一个特例。

那么如何求上述两个点基问题呢?在讲算法之前,我们还要介绍一个概念:

设$[S_i]$是有向图 G 的一个强连通分支，如果在 G 中，不存在终点属于$[S_i]$而起点不属于$[S_i]$的弧，就称$[S_i]$为最高的强连通分支。

用这个定义判断篮球队通讯图的 8 个强连通分支（图 4-11(*b*)），$[S_1]$、$[S_4]$和$[S_8]$是最高的，而其它 5 个都不是最高的（见图 4-13）。

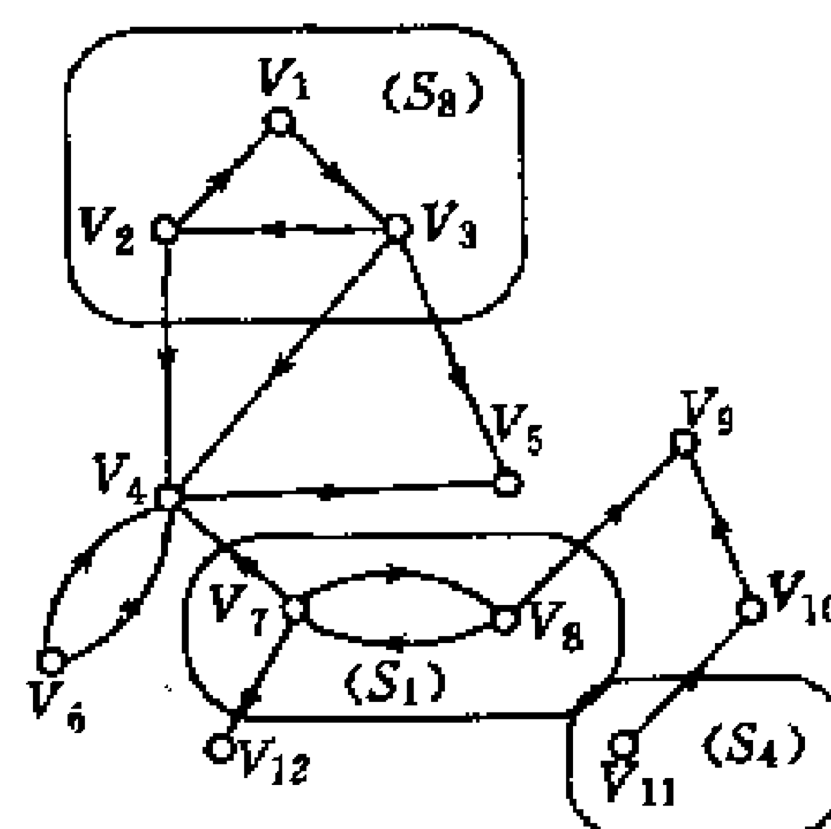


图 4-13

在什么意义上讲，具有上述特征的强连通分支$[S_i]$是最高的呢？由最高强连通分支的定义可以看出，该分支与图 G 其它部分相连的所有弧是向外伸展的，即 V_j 属于$[S_i]$而 V_k 不属于$[S_i]$，那末 V_j 不会是 V_k 的后代。针对教练下达通知的问题来说，如果 V_j 是一个属于最高强连通分支$[S_i]$的队员，那么任何不属于$[S_i]$的队员 V_k 是无法通知到他的。因此教练至少要直接通知$[S_i]$中的一个队员。

这一下就把求最小点基的算法由来说得再清楚不过了。如果顶点集合 B 是一个点基，那么每个最高强连通分支的$[S_i]$至少有一个顶点要属于 B。最小点基 B 中的顶点个数即为图 G 中最高强连通分支的分支个数。从每个最高强连通分支中任取一个顶点就可以组成这个最小点基 B。

至于求权最小点基的算法，从表面上看似乎比求最小点基复杂，其实不然，这两个问题的复杂程度是差不多的。要求一个权最小的点基，也需要进行 3 个步骤：

步骤 1　找出图 $G=(V,A)$的所有强连通分支；

步骤 2　从强连通分支中找出所有最高的强连通分支；

步骤 3　从每一个最高的强连通分支中取一个权最小的顶点，组成的顶点集 B 就是一个 G 的权最小的点基。

其中步骤 1,2 和求最小点基一样，而步骤 3 中，两个点基问题“从每一个最高强连通分支中取一个顶点”的方式不一样，求最小点基是“任取”，权最小点基是要求取的顶点权最小。

二、求最小点基的算法

求最小点基问题必须求极大强连通子图。4.4 节中给出了 *dfs* 算法求解的过程。求极大强连通子图在图论中有很大应用价值。为了便于读者今后更好地开发这一算法，我们在这里给出第二种解法。在求最小点基程序中，我们使用新方法求解。

从极大强连通分支的概念出发，很容易得出一种解法：

首先任取一个顶点 V_i，然后把所有与 V_i 互相可达的顶点 V_j 都找出来，这些顶点的集合 S_1（注意 V_i 也属于 S_1）生成的子图$[S_1]$（就是 S_1 和所有起点和终点都属于 S_1 的弧组成的那个子图）就是包含 V_i 的强连通分支，然后再取一个不属于 S_1 的顶点 V_j，再求出与 V_j 互相可达的顶点集合 S_2，再生成一个强连通分支$[S_2]$，接下去再取一个既不属于 S_1 又不属于 S_2 的顶点 V_k……，最后就可以把所有强连通分支都求出来了，图的每一个顶点都属于一个强连通分支。

那么，如何从一个不属于目前任何强连通分支的顶点 V_i 出发，扩展出下一个强连通分支$[S_k]$呢？

如果存在一个顶点 V_j 与 V_i 互相可达，那么 V_j 即是 V_i 的后代（因为 V_i 可达 V_j），又是 V_i 的前代（因为 V_j 可达 V_i）。因此，要求所有与 V_i 互相可达的顶点集合 S_k，只要先求出 V_i 的所有后代的集合 R，再求出 V_i 的所有前代的集合 P。然后，找出所有既属于 R 又属于 P 的顶点，这些顶点就组成了集合 S_k。

为了求出 V_i 所有后代的集合 R（所有前代的集合 P），也可以采用一种标号法，在这种标号法中，一旦能确定一个顶点是 V_i 的后代（前代），就可给它一个标号＋（－）。具体计算时，首先给 V_i 以标号＋（－），这是因为 V_i 可达自己，因此 V_i 本身也是 V_i 的后代（前代），然后逐步扩大已标号点的范围。办法是：如果发现一个顶点 V_k 已标号，说明它是 V_i 的后代（前代），即存在一条从 V_i 到 V_k（从 V_k 到 V_i）的有向路。而与 V_k 相连接的是一条以 V_k 为起点的弧$\langle k,j\rangle$（以 V_k 为终点的弧$\langle j,k\rangle$），这时如果 V_j 还没有得到标号，就可以给 V_j 以标号＋（－）。因为从 V_i 到 V_k 的有向路（从 V_k 到 V_i 的有向路）上接上弧$\langle k,j\rangle$（$\langle j,k\rangle$）就是一条 V_i 到 V_j 的有向路（V_j 到 V_i 的有向路）。因此 V_j 也是 V_i 的后代（前代）。这样不断地扩大已标号顶点的范围，直至无法扩大为止，这时所有已标号的顶点就恰好是 V_i 的后代集合 R（前代集合 P）了。

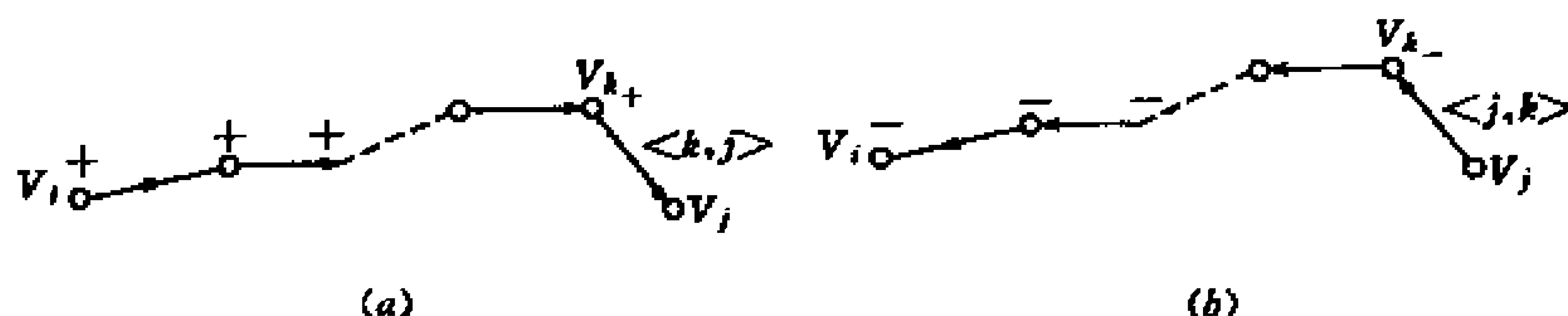


图 4-14

当一个图中顶点和弧很多时（见图 4-14），为了使标号过程有条不紊，从而避免重复，通常在考虑一个有标号的顶点 V_k 时，应该把所有以 V_k 为尾的弧$\langle k,j\rangle$（V_k 为头的弧$\langle j,k\rangle$）都考察一遍。如果$\langle k,j\rangle$的终点 V_j（$\langle j,k\rangle$的起点 V_j）还没有标号，就给它标上号。考察完以后，就可以认为顶点 V_k 已经“检查”过了，今后就不必再考虑它了，因为它再也不会使别的顶点得到标号了。接下来再取一个已标号而没有“检查”过的顶点来考虑……，如果发现所有标号的顶点都已经被检查过了，那么很显然，已经不可能再扩大已标号点的范围了。

在讲解标号法的过程中，( )里的内容对应于求前代集合 P。

把上面讲的总结起来，就可以得到下面的求顶点 V_i 的所有后代的集合 R 和所有前代的集合 P 的计算方法了：

1. 求 V_i 所有后代的集合 R

步骤 1. 给 V_i 以标号＋，V_i 是已标号而未检查的顶点；

步骤 2. 看看是否有已标号而未检查的顶点，如果没有，计算结束，所有已标号的顶点组成集合 R。如果有，任取一个这样的顶点 V_k，转入步骤 3；

步骤 3. 找出以 V_k 为起点的所有弧$\langle k,j\rangle$，一条一条地考虑这些弧，如果$\langle k,j\rangle$的

终点 V_j 没有标号，就给 V_j 以标号＋，V_j 成为已标号而未检查的顶点。这些弧考虑过以后，V_k 成为已检查过的，作一个检查标志，转回步骤 2。

2. 求顶点 V_i 的所有前代的集合 P

采用与求 R 集合完全相似的算法。步骤 1 和 2 是一样的。唯一的差别是步骤 3。本来在取定了 V_k 以后，考虑的是以 V_k 为起点的弧 $<k,j>$，现在则应该改为考虑以 V_k 为终点弧 $<j,k>$，如果这条弧的起点 V_j 还没有标号，就应该给它以标号。另外，为了在一张图上同时求 V_i 的所有前代与后代，那么在求前代时，可以用‘－’作为标号。

很显然，在求出集合 R 和 P 以后，既标有＋又标有－的顶点就是与 V_i 互相可达的顶点了，这些顶点组成了一个强连通分支的顶点集合 S_k。

三、求最小点基的程序

```
program dian_ji;
  uses crt;
  const maxn = 100;
  type  list       = array[1..maxn] of boolean;
  var   tu         : array[1..maxn] of list;  { 图的邻接矩阵 }
        s          : list;             { 记录顶点 i 是否已属于某强连通分支 }
        n          : integer;          { 顶点数 }
  procedure init;
    var i,j,e,k : integer;
      begin
        clrscr;
        repeat write('n='); { 输入顶点数 }
              readln(n);
        until (n>0) and (n<=maxn);
        for i:=1 to n do  { 邻接矩阵初始化 }
          for j:=1 to n do
            tu[i,j]:=false;
        write('e=');
        readln(e);
        for k:=1 to e do { 读入邻接矩阵 }
          begin
            read(i);
            readln(j);
            tu[i,j]:=true;
          end;
      end;
  procedure make_s_i(x:integer;var v_i:list); {构造顶点 x 所属的强连通分支 v_i }
    var  shun,                         { x 的所有后代的集合 }
         ni,                           { x 的所有前代的集合 }
         b        : list;              { 记录当前顶点是否检查过 }
         more     : boolean;           { 前代(后代)集合未产生标志 }
         i,j      : integer;
    begin
      for i:=1 to n do { 后代集合,前代集合以及检查标志初始化 }
        begin
```

```
          shun[i] := false;
          ni[i] := false;
          b[i] := false;
        end;
      shun[x] := true; ni[x] := true; {设 x 在所属的强连通分支上}
      repeat more := false; { 求出 x 的所有后代集合,这些后代设检查标志 }
            for i := 1 to n do
              if (shun[i]) and (not b[i])
              then begin
                      for j := 1 to n do
                        if (tu[i,j]) and (not shun[j])
                        then shun[j] := true;
                      more := true;
                      b[i] := true;
                    end;
      until not more;
      for i := 1 to n do { 检查标志初始化 }
        b[i] := false;
      repeat more := false; { 求出 x 的所有前代集合,这些前代设检查标志 }
            for i := 1 to n do
              if ni[i] and (not b[i])
              then begin
                      for j := 1 to n do
                        if (tu[j,i]) and (not ni[j])
                          then ni[j] := true;
                        more := true;
                        b[i] := true;
                    end;
      until not more;
      for i := 1 to n do { 求 x 所属的强连通分支上的所有顶点 }
        v_i[i] := shun[i] and ni[i];
    end;
function highest(s_i : list) : boolean;
  { 搜索所有弧,若不存在终点属于 s_i 而起点不属于 s_i 的弧,        }
  { 则强连通分支 s_i 是最高强连通分支,返回 true;否则返回 false }
  var i,j  :  integer;
    begin
      highest := false;
      for i := 1 to n do
        for j := 1 to n do
          if (tu[i,j]) and (not s_i[i]) and (s_i[j])
          then exit;
      highest := true;
    end;
procedure main;
  var s_i  :  list;              { 当前顶点 i 属于的强连通分支 }
      i,j  :  integer;
    begin
      for i := 1 to n do { 设所有的顶点不在强连通分支上 }
```

```
      s[i] := false;
    for i := 1 to n do
      if not s[i] then { 顶点 i 目前不属于任何强连通分支 }
        begin
          make_s_i(i,s_i);      { 构造顶点 i 所属的强连通分支 s_i }
          if highest(s_i)              { 若 s_i 是最高强连通分支 }
            then writeln('take ',i); { 则 i 属于最小点基,输出 i }
          for j := 1 to n do
            s[j] := s[j] or s_i[j];
          { s_i 归入已生成的强连通分支 }

        end;
  end;
begin
  init; { 输入图 }
  main; { 计算和输出最小点基 }
end.
```

4.6 可靠通讯网的构作

一、问题及其分析

构作一个具有 n 个通讯站的有线通讯网,使得敌人至少炸坏我 m 个($m<n$)通讯站后,才能中断其余的通讯站的彼此通话。显然有两个要求是必要的:一是不怕被敌人炸坏站的数目要多,二是整个造价要低。这个实际问题的数学模型如下:

G 是加权连通无向图,m 是给定的自然数,求 G 的最小权的 m 连通生成子图。当 $m=1$ 时,它就是 3.1 节所讲的求无向图的最小生成树;当 $m>1$ 时是尚未解决的难题之一。

我们将原来的问题作两条限制后就可以解决:

1. G 是完全图;

2. 每边的权皆为 1。

因为当每条通讯线(无论其长短)都造价一样时,要满足第 1 个要求,最理想的情况无非是原来每两个通讯站之间皆能彼此通话。但这样做造价太高,必须在满足第 1 个要求的前提下尽可能多地减小通讯线数,以达到两个要求同时满足。换句图论的话说,当 G 为 n 个顶点的完全图且每边权皆为 1 时,求一个 G 的边数最小的 m 连通子图。

令 $f(m,n)$ 表示 n 个顶点的 m 连通子图中边数的最小值($m<n$)。由 G 的 n 个顶点的次数和 $=2\times G$ 的边数,$K\leqslant K'\leqslant G$ 的顶点度数的最小值(K 和 K' 为 G 的连通度和边连通度),得出:

$$f(m,n)\geqslant \{m\ n/2\}$$

由上可见,构作一个由最少边数连接 n 个顶点的 m 连通图。这个图的边数恰为 $\{mn/2\}$,此图记成 $H_{m,n}$。

我们根据 m 和 n 的奇偶性作图。

1. 当 m 是偶数($m=2r$)

$H_{2r,n}$ 以 $\{0,1,\cdots,n-1\}$ 为顶点集合。当 $i-r\leqslant j\leqslant i+r$ 时,在顶点 i 与 j 之间连一条

边。这里加法在 mod n 意义下进行。例如 4-15(a)是构作的 $H_{4,8}$图。

2. 当 m 是奇数($m=2r+1$)，先构作 $H_{2r,n}$图，然后再根据 n 的奇偶性添边：

n 是偶数：

对 $1\leqslant i\leqslant n/2$ 的 i，在 i 与 $i+n/2$ 间加上一条边得 $H_{2r+1,n}$。例如 4-15(b)是构作的 $H_{5,8}$图。

n 是奇数：

在顶点 0 与顶点$(n-1)/2$、顶点 0 与顶点$(n+1)/2$ 之间加上两条边，在顶点 i 与顶点 $i+(n+1)/2$ 间加上边，其中 $1\leqslant i\leqslant(n-1)/2$，则得到 $H_{2r+1,n}$。例如 4-15(c)是构作的 $H_{5,9}$图。

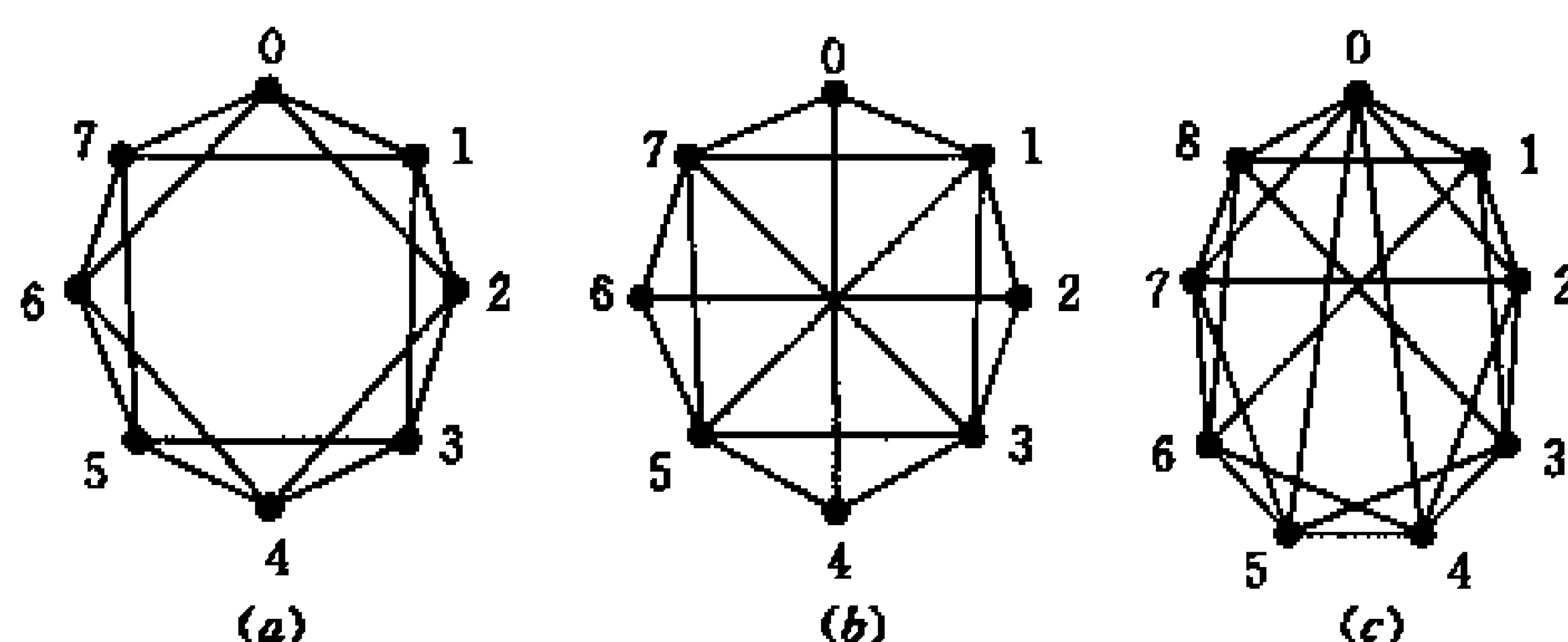


图 4-15

二、求 $H_{m,n}$图的程序

下面是求 $H_{m,n}$图的程序。

```
program harary_p53;


  var
    m,n,i,j       :  integer;

  procedure init;
    begin
      repeat
        write('m n = ');readln(m,n) { 读入连通数和顶点数 }
      until (m<n)
    end;

  function change(a,n : integer) : integer; { 返回 a 除以 n 的余数 }
    begin
      if a<0
         then change : =(n+a) mod n { 逆时针取余 }
         else change : =a mod n      { 顺时针取余 }
    end;

  procedure A(r,n : integer);
```

```
{ 以{0..n-1}为顶点集合,当 i-r≤j(在 mod n 定义下进行)≤i+r 时 }
{ 在顶点 i 与 j 之间连一条边 }
  begin
    for i:=0 to n-1 do
      for j:=i-r to i+r do
        if change(j,n)>i then writeln(i:2,'——',change(j,n):2)
  end;

procedure pp;
  begin
    if not odd(m) { m 是偶数,构作 H2r,n}
       then A(m div 2,n)
       else if not odd(n) { m 是奇数,n 是偶数 }
               then begin
                      A((m-1) div 2,n); { 先构作 H2r,n}
                      for i:=1 to n div 2 do writeln(i:2,'——',i+n div 2);
                      { 在 i 与 i+n/2 间加一条边(1≤i≤n/2),得 H2r+1,n}
                    end
               else begin { m 是奇数,n 是奇数 }
                      A((m-1) div 2,n); { 先构作 H2r,n}
              { 在顶点 0 与顶点(n-1)/2 之间,顶点 0 与顶点(n+1)/2 之间加两边 }
                      writeln(0:2,'——',(n-1) div 2:2);
                      writeln(0:2,'——',(n+1) div 2:2);
                      for i:=1 to (n-1) div 2-1 do
                      { 连接顶点 i 与顶点 i+(n+1)/2(1≤i≤(n-1)/2),得 H2r+1,n}
                        writeln(i:2,'——',i+(n+1) div 2);
                    end
  end;

begin
  init; { 读入连通数 m 和顶点数 n }
  pp    { 构作可靠通讯网并输出 }
end.
```

第五章 支配集与独立集

5.1 求支配集

一、问题及其分析

要在 $V_1,V_2,\cdots,V_n$ 这 N 个城镇建立一个通迅系统，为此从这 n 个城镇中选定几座城镇，在那里建立中心台站，要求它们与其它各城镇相邻，同时为降低造价，要使中心台站数目最少，有时还会提其它要求。例如在造价最低的条件下，需要造两套(或更多套)通讯中心，以备一套出故障时启用另一套。

例如图 5-1 看作是一个通讯系统。$V_1,V_2,\cdots,V_6$ 是一些城镇，仅当两城之间有直通通讯线时，相应的两顶点连一条边。在讲这个问题的数学模型之前，先讲一个定义：

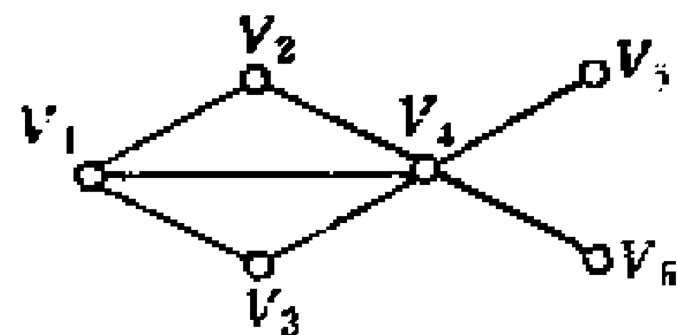


图 5-1

D 是图 G 的一个顶点子集，对于 G 的任一顶点 U，要么 U 是 D 集合的一个顶点元素，要么与 D 中的一个顶点相邻，那么 D 称为图 G 的一个支配集。若在 D 集中去除任何元素后 D 不再是支配集，则支配集 D 是极小支配集。称 G 的所有支配集中顶点个数最少的支配集为最小支配集 D_0，记 $\gamma(G)=D_0$ 中的顶点个数，称作 G 的支配数。

由上述定义可知，凡最小支配集一定是极小支配集；任何一个支配集以一个极小支配集为其子集；G 所含的极小支配集可能有两个以上，而且其顶点数也可以不一致，但支配数 $\gamma(G)$ 是唯一的。

显然中心台站的选址问题，实际上是求一个能与 G 中其它顶点相邻的顶点集合，且这个顶点集合所含的顶点元素必须最少，即求图的最小支配集。若建两套，则从一切极小支配集 $D_1,D_2,\cdots,D_d$ 中选取 D_m 和 D_n，使得 $D_m\cap D_n=\varphi$(φ 为支配数，愈小愈好)，即其中一套出故障时不影响另一套工作。并且 $|D_m\cup D_n|=\min\{|D_i|+|D_j|\mid 1\leqslant i,j\leqslant d,D_i\cap D_j=\varphi\}$，即两套互不干扰的通讯中心所选定的城镇数最少。

例如在图 5-1 所示的 6 个城镇中建中心台站，方案有 $\{V_1,V_5\}$，$\{V_1,V_6\}$，$\{V_4\}$，$\{V_2,V_3,V_5\}$，$\{V_2,V_3,V_6\}$。这些顶点集合为极小支配集。若建一套通讯中心，只需建立在 V_4 城；若建两套，则 V_4 建一套，$\{V_1,V_5\}$ 或者 $\{V_1,V_6\}$ 建第 2 套。

有许多实际问题可以转化成上述这种数学模型来处理。

支配集有哪些性质呢？

1. 若 G 中无零次顶点($d(v)=0$)，则存在一个支配集 D，使得 G 中除 D 外的所有顶点也组成一个支配集；

2. 若 G 中无零次顶点($d(v)=0$)，D_1 为极小支配集，则 G 中除 D_1 外的所有顶点组成一个支配集。

求所有极小支配集的计算，包括 5.2 节中的求所有极小覆盖集的计算都是采用逻辑

运算的。

设 X,Y,Z 三条指令：

规定："要么执行 X，要么执行 Y" 记作 $X+Y$（和运算）

"X 与 Y 同时执行" 记作 XY （与运算）

上述逻辑运算有以下运算定律：

1. 交换律

   $X+Y=Y+X$；$XY=YX$

2. 结合律

   $(X+Y)+Z=X+(Y+Z)$；$(XY)Z=X(YZ)$

3. 分配律

   $X(Y+Z)=XY+XZ$；$(Y+Z)X=XY+XZ$

4. 吸收律

   $X+X=X$；$XX=X$；$X+XY=X$

上述定律尤其是吸收律，在求所有极小支配集和极小覆盖集的两个公式中用处颇大。求所有极小支配集的公式：

$$\varphi(V_1,V_2,\cdots,V_n)=\prod_{i=1}^{n}\left(V_i+\sum_{U\in N(V_i)}U\right)$$

一个顶点同与它相邻的所有顶点进行加法运算组成一个因子项，几个因子项再连乘。连乘过程中根据上述运算规律展开成积之和形式。每一积项给出一个极小支配集，所有积项给出了一切极小支配集，其中最小者为最小支配集。

例如求图 5-1 通讯网的一切极小支配集及支配数。

$\varphi(V_1,V_2,V_3,V_4,V_5,V_6)$

$=(V_1+V_2+V_3+V_4)(V_2+V_1+V_4)(V_3+V_1+V_4)(V_4+V_1+V_2+V_3+V_5+V_6)$

$(V_5+V_4+V_6)(V_6+V_4+V_5)$

$=(1+2+3+4)(2+1+4)(3+1+4)(4+1+2+3+5+6)(5+4+6)(6+4+5)$

$=15+16+4+235+236$

故得所有极小支配集如下：

$\{V_1,V_5\},\{V_1,V_6\},\{V_4\},\{V_2,V_3,V_5\},\{V_2,V_3,V_6\}$

$\gamma(G)=1$

二、求极小支配集和支配数的程序

下面根据公式和运算定律，给出求所有极小支配集和支配数的程序。

```
Program ji_xiao_zhi_pei_ji;

const
  maxn        =  30;

type
  ghtype      =  array [1..maxn,1..maxn] of integer;
```

```
  settype        =   set of 1..maxn;
  ltype          =   array [1..maxn] of settype;

var
  g              :   ghtype;     { 邻接矩阵 }
  n,l            :   integer;    { 顶点数,极小支配集的个数 }
  f              :   text;       { 文件变量 }
  lt             :   ltype;      { 极小支配集 }

procedure read_graph;
  var
    str : string;
    i,j : integer;
  begin
    write('Graph file = '); { 输入文件名,并与文件变量连接 }
    readln(str);
    assign(f,str);
    reset(f);
    readln(f,n);          { 输入顶点数 }
    for i := 1 to n do
      for j := 1 to n do read(f,g[i,j]); { 输入图的邻接矩阵 }
    close(f);
  end;

procedure reduce(s : settype);
{ 根据吸收律求 s+lt[1]+lt[2]+…+lt[l]的结果 }
  var
    i,j : integer;
  begin
    i := 1;
    while i<=l do { 检查所有支配集 }
      begin
        if s * lt[i]=lt[i] then exit;
        { 若第 i 个支配集是 s 的子集,则退出过程 }
        if s * lt[i]=s
        { 若 s 是第 i 个支配集的子集,则删除该极小支配集 }
          then begin
                 for j := i+1 to l do lt[j-1] := lt[j];
                 dec(l)
               end
          else inc(i)
        { 否则检查下一个支配集,即删除所有含 s 子集的支配集 }
        { 直至某支配集作为 s 的子集或所有支配集与 s 非子集关系为止 }
      end;
    inc(l);lt[l] := s { s 作为最后一个支配集 }
  end;

procedure think;
  var
```

```
    tl,i,j,k : integer;
    t : ltype;
  begin
    l : =0;
    for i : =1 to n do if (i=1) or (g[1,i]>0) then reduce([i]);
    { 建立 v1+ ∑(u∈N(v1)) u,各项存入 lt }
    for i : =2 to n do
      begin
        t : =lt;tl : =l; { 暂存所有支配集 lt 和支配集个数 }
        l : =0;
        for j : =1 to n do
        { 求(v2+ ∑(u∈N(v2)) u) (v3+ ∑(u∈N(v3)) u)…(vi+ ∑(u∈N(vi)) u)的所有乘积项 lt[1]…lt[l] }
          if (i=j) or (g[i,j]>0)
            then for k : =1 to tl do
                  reduce(t[k]+[j])
      end;
    lt : =t;l : =tl { 求出 L 个极小支配集 lt }
  end;

procedure print; { 打印 l 个极小支配集中的所有元素 }
  var
    i,j : integer;
  begin
    for i : =1 to l do
      begin
        write(i : 3);
        for j : =1 to n do if j in lt[i] then write(j : 3);
        writeln
      end
  end;

begin
  read_graph;  { 输入图 }
  think;       { 计算极小支配集 }
  print        { 输出结果 }
end.
```

5.2 求独立集

一、问题及其分析

$S=\{S_1,S_2,\cdots,S_n\}$为信息传送的基本信号集合。

可以把信号 S_i 设想成汉字或拉丁字母。统计规律表明哪些信号与哪些信号易于发生错乱。例如输入 S_i,输出应该是 S_i^* ($1\leqslant i\leqslant 5$)。但由于干扰发生了错乱,S_1 可能和 S_2 错乱,S_1 还可能和 S_5 错乱等。例如已知错乱可能性如图 5-2(a)所示。

为了确切地从输出信号得知输入信号,我们不能选用 $S_1,S_2,\cdots,S_n$ 中的每一信号,只

能从中选一部分用于输出。那么选哪一部分信号作输入源呢？我们作另外一张图，顶点表示信号，若信号源 S_i 可能输出 S_j，则在 S_i 与 S_j 之间连一条边，如图 5-2(b)。为了使可用于输出的信号最多，我们在图 5-2(b)上求这样一个顶点集合 I，I 中任两个顶点不相邻且这些独立的顶点数最多。例如可以选 $\{S_1, S_3\}$ 作为输出信号；或选 $\{S_1, S_4\}$ 或 $\{S_2, S_4\}$ 或 $\{S_2, S_5\}$ 或 $\{S_3, S_5\}$ 做输出信号。这个数学模型就是所谓的求最大独立集问题。

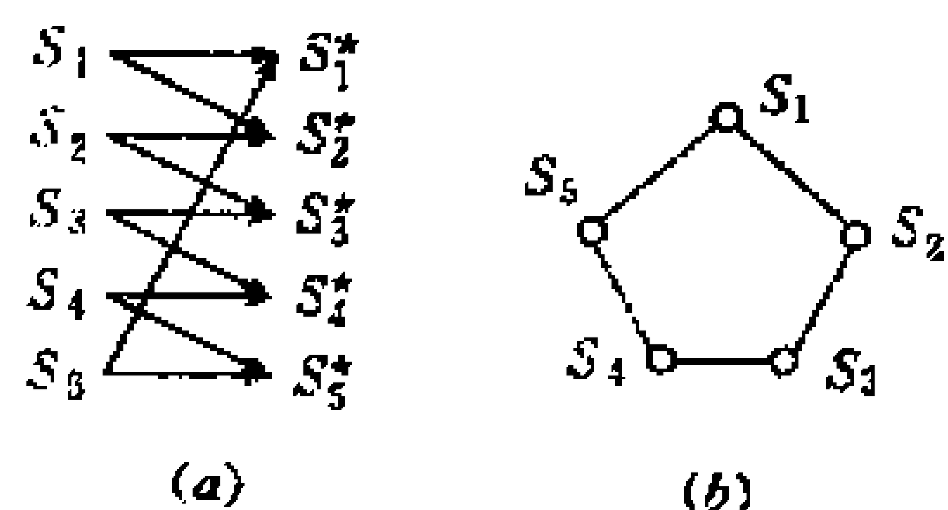


图 5-2

在实际生活中，求最大独立集的应用实例很多，如 8 皇后问题。我们将棋盘的每个方格看作一个顶点，放置的皇后所在格与她能攻击的格看成是有边连接，从而组成 64 个顶点的图。若要设计一个算法，放置尽可能少的皇后，使她们控制棋盘上全部格，这显然是求最小支配集问题；再设计一个算法，要放置尽可能多的皇后，而相互不攻击地共处．这个问题就是求最大独立集问题了。

在讲求最大独立集的定义之前，我们先引出与独立集密切相关的覆盖集的概念：K 是 G 图的一个顶点子集且 G 的每一边至少有一个端点属于 K，则称 K 是 G 的一个覆盖。这里覆盖一词的含义是顶点覆盖全体边。若在 K 集中去除任一顶点后将不再是覆盖，则称 K 为极小覆盖。在图 G 的所有极小覆盖集中含顶点数最少的极小覆盖称作最小覆盖。用 $\alpha(G)$ 表示最小覆盖的顶点数，又称 G 的覆盖数。

例如图 5-3 中的黑色顶点是一个极小覆盖，同时也是一个最小覆盖，$\alpha(G)=4$，$K=\{V_1, V_3, V_4, V_6\}$。下面给出独立集的概念：

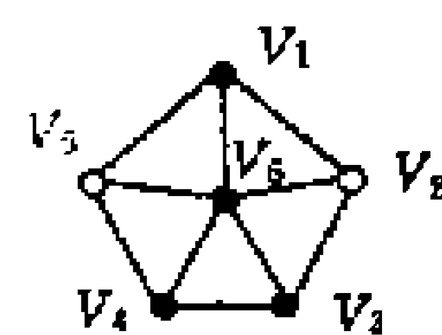


图 5-3

I 是 G 的一个顶点子集，I 中任两个顶点不相邻，则称 I 是图 G 的一个独立集。若独立集 I 中增加任一个除 I 集外的顶点后 I 不是独立集，则称 I 是极大独立集。在图 G 的所有极大独立集中含顶点数最多的极大独立集称作最大独立集。用 $\beta(G)$ 表示最大独立集的顶点数，又称 G 的独立数。

那么独立集又有什么性质呢？我们从独立集、覆盖集、支配集之间关系中论述：

1. 一个独立集是极大独立集，当且仅当它是一个支配集；

2. I 是独立集，当且仅当 G 中除 I 集外的所有顶点是一个覆盖集；

I 是极大(最大)独立集，当且仅当 G 中除 I 集外的所有顶点是一个极小(最小)覆盖集，即

$$\alpha(G) + \beta(G) = G \text{ 的顶点数}$$

3. 极大独立集必为极小支配集；但是极小支配集未必是极大独立集。

例如一个含边数为 4 的圈上的两个相邻顶点是极小支配集，但不能为极大独立集，连独立集也不是。

例如图 5-3 中 $\{V_1, V_3, V_4, V_6\}$ 是一个最小覆盖集，$V-K=\{V_2, V_5\}$ 是一个最大独立集同时又是一个极小支配集。注意图 5-3 中的最小支配集是 $\{V_6\}$ 不是独立集。

由于极大独立集与极小覆盖集有互补性，我们这里仅给出求所有极小覆盖集的计算

公式，读者可以由此推出极大独立集：

$$\varphi(V_1,V_2,\cdots,V_n)=\prod_{i=1}^{n}\left(V_i+\prod_{U\in N(V_i)}U\right)$$

某顶点的所有相邻顶点进行积运算后再与该顶点进行和运算，组成一个因子项。几个因子项连乘，并根据逻辑运算定律展开成积之和形式。每一积项给出一个极小覆盖集，所有积项给出了一切极小覆盖集，其中最小者为最小覆盖集。

例如求图 5-4 中的一切极小覆盖集和覆盖数 $\alpha(G)$以及一切极大独立集和独立数 $\beta(G)$。

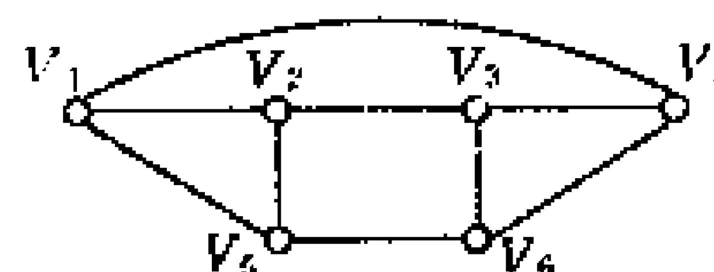


图 5-4

$\varphi(V_1,V_2,\cdots,V_6)$

$=(1+246)(2+136)(3+246)(4+135)(5+346)(6+125)$

$=2456+1356+1346+2346+1245+1235$

即得一切极小覆盖集如下

$\{V_2,V_4,V_5,V_6\},\{V_1,V_3,V_5,V_6\},$

$\{V_1,V_3,V_4,V_6\},\{V_2,V_3,V_4,V_6\},$

$\{V_1,V_2,V_4,V_5\},\{V_1,V_2,V_3,V_5\}$。

$\alpha(G)=4$

由于 G 的除极小覆盖集外的顶点可组成极大独立集，由此可以得出一切极大独立集为

$V-\{V_2,V_4,V_5,V_6\}=\{V_1,V_3\}$；

$V-\{V_1,V_3,V_5,V_6\}=\{V_2,V_4\}$；

$V-\{V_1,V_3,V_4,V_6\}=\{V_2,V_5\}$；

$V-\{V_2,V_3,V_4,V_6\}=\{V_1,V_5\}$；

$V-\{V_1,V_2,V_4,V_5\}=\{V_3,V_6\}$；

$V-\{V_1,V_2,V_3,V_5\}=\{V_4,V_6\}$。

$\beta(G)=2$

可见求最大独立集，首要的是求极小覆盖集。

二、求所有极小覆盖集的程序

下面我们给出求所有极小覆盖集的程序。

```
program ji_xiao_fu_gai_ji;

const
  maxn         =   30;

type
  ghtype       =   array [1..maxn,1..maxn] of integer;
  settype      =   set of 1..maxn;
  ltype        =   array [1..maxn] of settype;

var
  g            :   ghtype;      { 邻接矩阵 }
```

```
    n,l           :  integer;    { 顶点数,极小覆盖集个数 }
    f             :  text;       { 文件变量 }
    lt            :  ltype;      { 极小覆盖集 lt[i]—第 i 个极小覆盖集 1≤i≤l }
  procedure read_graph;
    var
      str : string;
      i,j : integer;
    begin
      write('Graph file = '); {输入文件名并与文件变量连接 }
      readln(str);
      assign(f,str);
      reset(f);
      readln(f,n); { 输入顶点数和图矩阵 }
      for i : =1 to n do
        for j : =1 to n do read(f,g[i,j]);
      close(f);
    end;

  procedure reduce(s : settype); { 根据吸收律求 s+lt[1]+…+lt[l]的结果 }
    var
      i,j : integer;
    begin
      i : =1;
      while i<=l do
        begin
          if s * lt[i]=lt[i] then exit;
          if s * lt[i]=s
            then begin
                   for j : =i+1 to l do lt[j-1] : =lt[j];
                   dec(l)
                 end
            else inc(i)
        end;
      inc(l);lt[l] : =s
    end;

  procedure think;
    var
      tl,i,j,k : integer;
      t : ltype;
      s : settype;
    begin
      lt[1] : =[1];lt[2] : =[];l : =2;
      for i : =2 to n do if g[1,i]>0 then lt[2] : =lt[2]+[i]; { 建立 v1 + ∏ u (u∈N(v1)) }
        for i : =2 to n do { 求 ∏(i=1..n) (vi + ∏ u (u∈N(vi))) }
```

```
      begin
        t := lt; tl := l;
        l := 0; s := [];
        for j := 1 to n do if g[i,j]>0 then s := s+[j];
        { 求所有与顶点 i 相连的顶点集合 s }
        for k := 1 to tl do
          begin
          reduce(t[k]+s); reduce(t[k]+[i])
            { 根据分配律求(t[1]+…+t[l])(s+[i])的所有乘积项 }
          end
      end;
      lt := t; l := tl
  end;

procedure print;
  var
    i,j : integer;
  begin
    for i := 1 to l do { 打印每个乘积项(极小覆盖集)的元素 }
      begin
        write(i : 3);
        for j := 1 to n do if j in lt[i] then write(j : 3);
        writeln
      end
  end;

begin
  read_graph;  { 输入图 }
  think;       { 计算极小覆盖集 }
  print        { 输出结果 }
end.
```

第六章　网络流及其应用

6.1　求网络的最大流

许多系统包含了流量问题。例如，公路系统中有车辆流，控制系统有信息流，供水系统中有水流，金融系统中有现金流等等。

图 6-1(*a*)是联结产品产地 V_1 和销地 V_6 的交通网，每一弧(V_i,V_j)代表从 V_i 到 V_j 的运输线，产品经这条弧由 V_i 输送到 V_j，弧旁的数字表示这条运输线的最大通过能力(以后简称容量)。产品经过交通网从 V_1 输送到 V_6。现在要求制定一个运输方案，使 V_1 运到 V_6 的产品数量最多。

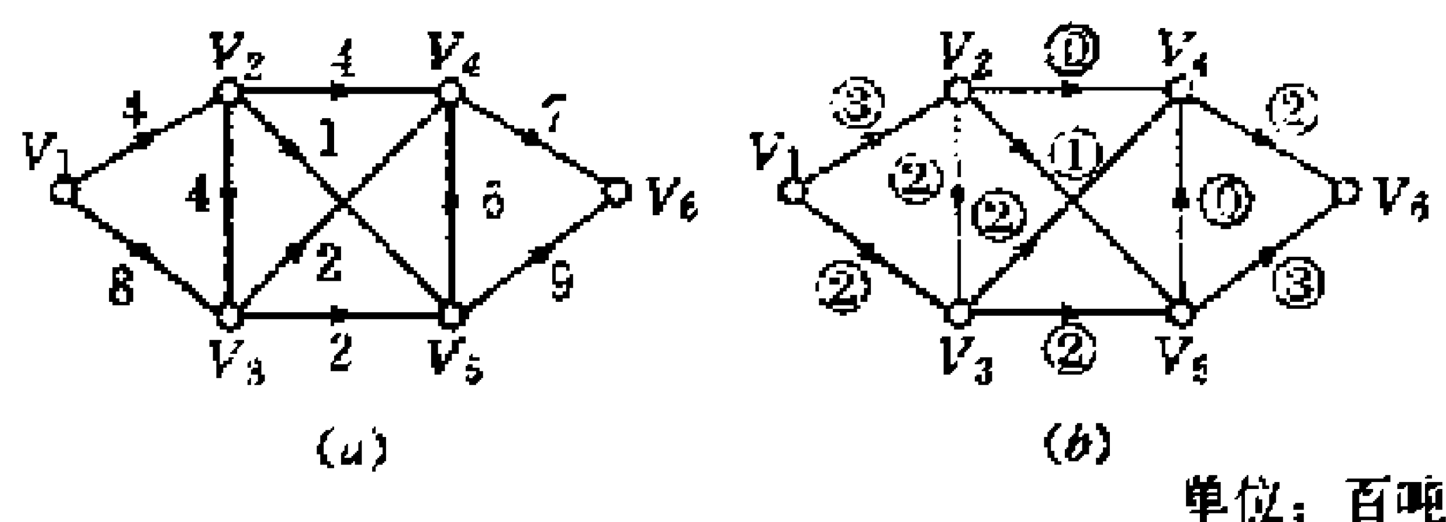


图　6-1

所谓运输方案，无非就是说，从产地运出多少物资，其中多少吨沿这一条路送到销地，多少吨沿那一条路送到销地等等。图 6-1(*b*)就给出了这样一种运输方案。2 百吨物资沿有向路 $P_1(V_1,V_2,V_3,V_4,V_6)$运到销地，2 百吨物资沿有向路 $P_2(V_1,V_2,V_3,V_5,V_6)$运到销地，1 百吨物资沿有向路 $P_3(V_1,V_2,V_5,V_6)$运到销地。每条弧旁的数字表示方案中每条运输线上的实际运输量。这些数必须满足一些条件，即：

1. 实际运输量不能是负的；

2. 每条弧的实际运输量不能大于该弧的容量；

3. 除了起点 V_1 和终点 V_6 外，对其它顶点来说，所有指向 V_i 的弧上的运输量的和应该等于所有从 V_i 出发的弧上的运输量的和。

这 3 个条件中的前两个条件的必要性是显然的，条件(3)也是不难理解的。因为，除了起点和终点以外，其余顶点都不过是一些"中转站"，它们既不能产生物质也不能囤积物质，因此运到它那里的物质是多少吨，从它那里运走的也应该是多少吨，不能多也不能少。显然，方案中的所有数满足了这 3 个条件，因此方案是可行的。但问题是，用图 6-1(*b*)的方案将 5 百吨的物质从 V_1 运到 V_6。在这个交通网上网的输送量是否还可以增多，或者说这个运输网中，从 V_1 到 V_6 的最大输送量是多少呢？本节将要研究类似这样的问题。

一、基本概念

1. 网络与流

网络 $D=(V,A,C)$

设 D 是一个简单有向图($D=(V,A)$)。在 V 中指定了一个顶点,称为源点(记为 V_s)和另一个顶点,称为汇点(记为 V_t),对于每一条弧 $(V_i,V_j)\in A$,对应有一个 $C_{ij}\geqslant 0$,称为弧的容量。通常我们就把这样的 D 叫做一个网络,记作 $D=(V,A,C)$。

网络流 F

所谓网络上的流是指定义在弧集合 A 上的一个函数 $F=\{F_{ij}\}$,并称 F_{ij} 为弧 (V_i,V_j) 上的流量。

例如图 6-1(a)即一个网络,指定 V_1 为源点,V_6 是汇点,其它的顶点是中间点。弧旁的数字为 C_{ij}。图 6-1(b)所示的运输方案,就可看作是这个网络上的一个流,每个弧上的运输量就是该弧上的流量,即:$F_{12}=3$,$F_{13}=2$,$F_{23}=2$,$F_{24}=0$,$F_{25}=1$,$F_{34}=2$,$F_{35}=2$,$F_{46}=2$,$F_{54}=0$,$F_{56}=3$。

2. 可行流与最大流

可行流(可行流的流量 $V(F)$)

满足下述条件的流 F 称为可行流:

(1) 容量限制条件:

对每一条弧 $(V_i,V_j)\in A$,$0\leqslant F_{ij}\leqslant C_{ij}$

(2) 平衡条件:

对于中间点:

流出量=流入量,即对每个 $i(i\neq s,t)$有

V_i 的流出量 $\sum\limits_{(v_i,V_j)\in A} F_{ij}$ $-$ V_i 的流入量 $\sum\limits_{(V_j,V_i)\in A} F_{ij}=0$

对于源点 S:

V_s 的流出量 $\sum\limits_{(V_s,V_j)\in A} F_{sj}$ $-$ V_s 的流入量 $\sum\limits_{(V_j,V_s)\in A} F_{js}=$ 源点的净输出量 $V(F)$。

对于汇点 T:

V_t 的流出量 $\sum\limits_{(V_t,V_j)\in A} F_{tj}$ $-$ V_t 的流入量 $\sum\limits_{(V_j,V_t)\in A} F_{jt}=(-1)*$ 汇点的净输入量 $V(F)$。

式中 $V(F)$称为这个可行流的流量,即源点的净输出量(或汇点的净输入量)。可行流总是存在的。例如:令所有弧的流量 $F_{ij}=0$,就得到一个其流量 $V(F)=0$ 的可行流(称为零流)。

最大流问题即求一个流$\{F_{ij}\}$,使其流量 $V(F)$达到最大,并且满足:

$$0\leqslant F_{ij}\leqslant C_{ij},(V_i,V_j)\in A$$

$$\sum F_{ij}-\sum F_{ji}=\begin{cases}V(F) & (i=s)\\ 0 & (i\neq s,t)\\ -V(F) & (i=t)\end{cases}$$

3. 可改进路 P

若给一个可行流 $F=(F_{ij})$，我们把网络中 $F_{ij}=C_{ij}$的弧称为饱和弧，$F_{ij}<C_{ij}$的弧称为非饱和弧，$F_{ij}=0$ 的弧称为零流弧，$F_{ij}>0$ 的弧称为非零流弧。

如图 6-1(*b*)中，(V_2,V_4)和(V_5,V_4)为零流，(V_3,V_6)和(V_3,V_5)是饱和弧，其它非零流弧为非饱和弧。

若 P 是网络中联结源点 V_s 和汇点 V_t 和一条路，我们定义路的方向是从 V_s 到 V_t，则路上的弧被分成两类：一类是弧的方向与路的方向一致，叫做前向弧。前向弧的全体记为 $P+$。另一类弧与路的方向相反，叫做后向弧。后向弧的全体记为 $P-$。

图 6-1(*a*)中，在路 $P=\{V_1,V_3,V_2,V_4,V_6\}$中

$$P+=\{(V_1,V_3),(V_2,V_4),(V_4,V_6)\}$$

$$P-=\{(V_3,V_2)\}$$

下面可引出可改进路 P 的定义：

设 F 是一个可行流，P 是从 V_s 到 V_t 的一条路，若 P 满足下列条件：

(1) 在 P 的所有前向弧(V_i,V_j)上，$0\leqslant F_{ij}<C_{ij}$，即 $P+$中的每一条弧是非饱和弧；

(2) 在 P 的所有后向弧(V_i,V_j)上，$0<F_{ij}\leqslant C_{ij}$，即 $P-$中的每一条弧是非零流弧。

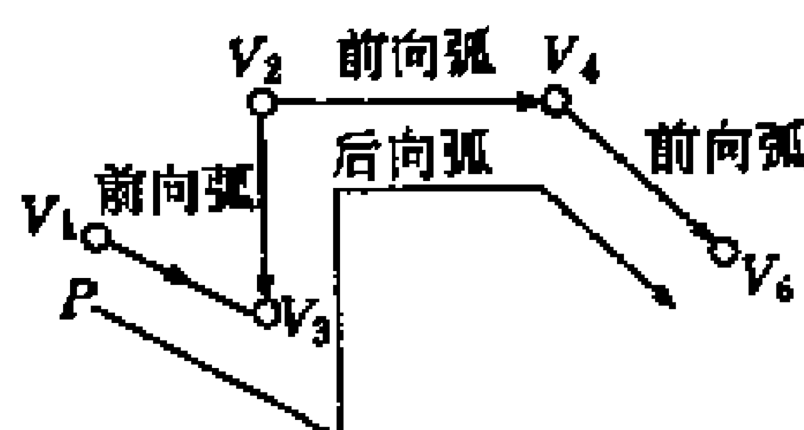


图 6-2

则称 P 为关于可行流 F 的一条可改进路。显然图 6-2给出的路 P 满足上述两个条件，是一条可改进路。那么，为什么将具有上述特征的路 P 叫做可改进路，原因是可以通过修正 P 路上所有流量 F_{ij}来把现有的可行流 F 改进成一个值更大的流 F_1。下面我们具体地给出一种方法，利用这种方法就可以把 F 改进成一个更好的流 F_1。这种方法是：

(1) 不属于可改进路 P 的弧$(V_i,V_j$上的流量一概不变，即 $F_{1ij}=F_{ij}$；

(2) 可改进路 P 上的所有弧(V_i,V_j)上的流量按下述规则变化：

在前向弧(V_i,V_j)上，　　$F_{1ij}=F_{ij}+a$

在后向弧(V_i,V_j)上，　　$F_{1ij}=F_{ij}-a$

a 称为可改进量，它应该按照下述原则确定：a 既要取得尽量大，又要使变化后的 F_{1ij} 仍满足可行流的两个条件——容量限制条件和平衡条件。不难看出，按照这个原则，a 即不能超过每条前向弧的 $C_{ij}-F_{ij}$，也不能超过每条后向弧的 F_{ij}。因此 a 应该等于前向弧上的 $C_{ij}-F_{ij}$与后向弧上的 F_{ij}的最小值。

图 6-2 给出一条可改进路 $P(V_1,V_3,V_2,V_4,V_6)$。现在就按照上面讲的方法将流 F 改进成一个更好的流。首先应该定出改进量 a，先看 P 的前向弧集合 $P+=\{(V_1,V_3),(V_2,V_4),(V_4,V_6)\}$：

$$C_{13}-F_{13}=8-2=6$$

$$C_{24}-F_{24}=4-1=3$$

$$C_{46}-F_{46}=7-3=4$$

再看 P 的后向弧集合 $P-=\{(V_2,V_3)\}$，在这条弧上 $F_{23}=2$。

总起来说，a 至多取 2，这样既可以使改进后的前向弧上的流量有所增加，又可以使改进后的后向弧上的流量在减少 a 之后不变负数。这个改进过程见图 6-3，改进后的流的值

增加为 8。

上面我们讲了在什么情况下，一个流可以改进和如何改进，但还有一点没有讲——怎样检查一个流是不是最大的。下面我们给出一个重要结论：

设 F 是网络 D 的一个流，如果不存在从 V_s 到 V_t 关于 F 可改进路 P，那么 F 一定是最大流。至此，我们不难得出求最大流的基本思路：

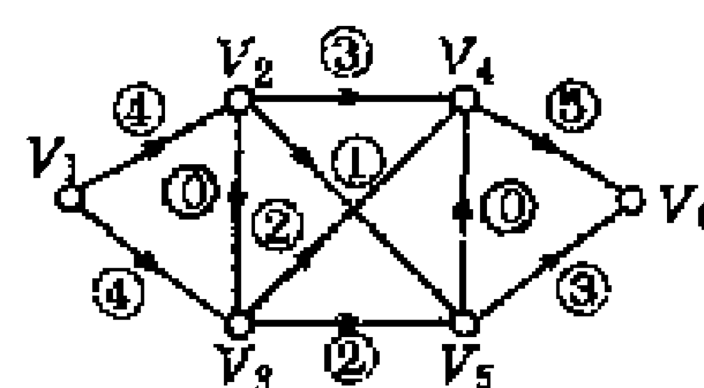
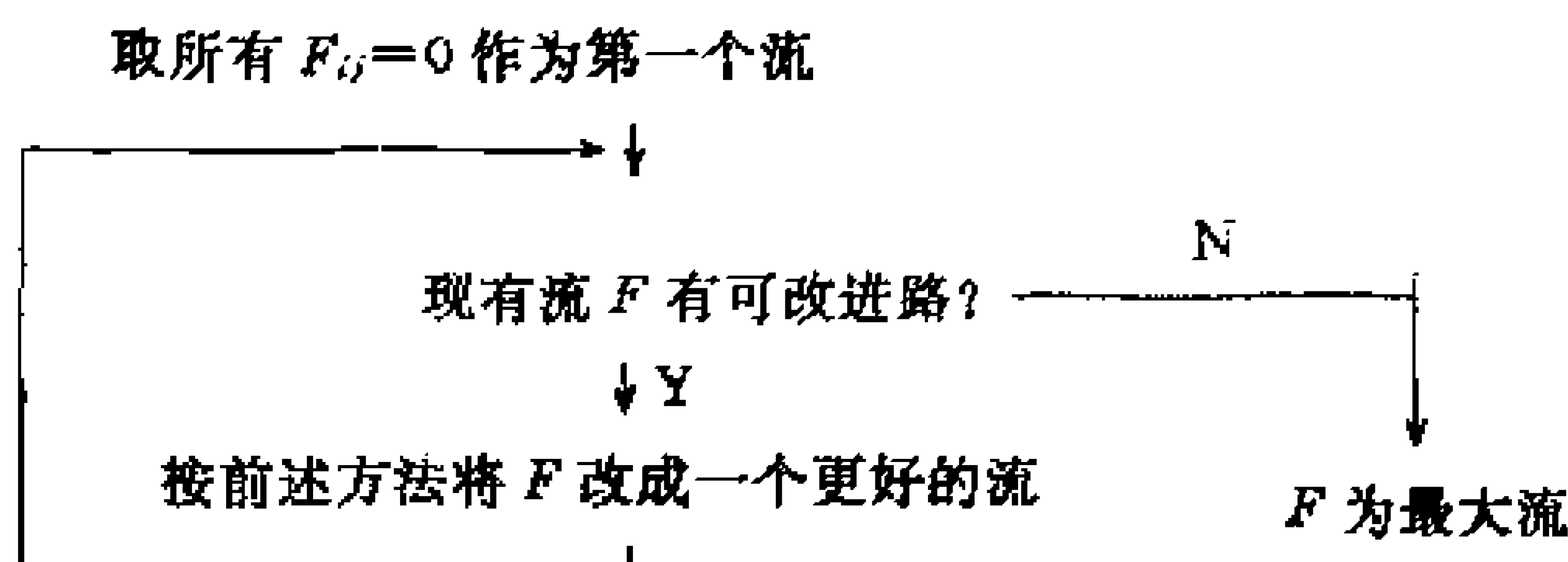


图 6-3



算法的关键和难点是如何寻找可改进路，特别是对于顶点数和弧数多的网络 D，寻找的效率至关重要。

是否能找到一种系统地寻找可改进路的方法，这种方法在存在可改进路时能很快地将它找出来，当不存在可改进路时能很快地告诉你“不用找了，肯定不会有”。下一节讲的标号法就是这样一种方法。

二、寻求最大流的标号法

从一个可行流出发（若网络中没有给定 F，则可以设 F 是零流），进入标号过程和调整过程。

1. 标号过程

在这个过程，网络中的顶点或者是标号点（又分为已检查和未检查两种），或者是未标号点。每个标号点的标号包含两个部分。第一标号指明它的标号从哪一顶点得到，以便找出可改进量；第二个标号是为确定可改进量 a 用的。

标号过程开始，总先给 V_s 标上$(0,+\infty)$，这时 V_s 是已标号而未检查的顶点，其余都是未标号点。一般，取一个标号而未检查的顶点 V_i，对一切未标号点 V_j：

(1) 若在弧(V_i,V_j)上 $F_{ij}<C_{ij}$，则给 V_j 标号$(V_i,L(V_j))$，这里 $L(V_j)=\min[L(V_i),C_{ij}-F_{ij}]$。这时顶点 V_j 成为标号而未检查的顶点。

(2) 若在弧(V_j,V_i)上 $F_{ij}>0$，则给 V_j 标号$(-V_i,L(V_j))$，这里 $L(V_j)=\min[L(V_i),F_{ij}]$。这时顶点 V_j 成为标号而未检查的顶点。

在 V_i 的全部相邻顶点都已标号后，V_i 成为标号而已检查过的顶点。重复上述步骤，一旦 V_t 被标上号，表明得到一条从 V_s 到 V_t 的可改进路 P，转入调整过程。

若所有标号都已检查过致使标号过程无法继续时，则算法结束，这时的可行流即最大流。

2. 调整过程

采用"倒向追踪"的方法，从 V_t 开始，利用标号点的第一个标号逐条弧地找出可改进路 P，并以 V_t 的第二个标号 $L(V_t)$ 作为改进量 a，改进 P 路上的流量。

例如设 V_t 的第一个标号为 V_k(或 $-V_k$)，则弧 (V_k,V_t)(或相应的 (V_t,V_k) 是 P 上的弧。接下来检查 V_k 的第一个标号，若为 V_i(或 $-V_i$)则找出 (V_i,V_k)(或相应的 (V_k,V_i))。再检查 V_i 的第一个标号，查到 V_s 为止。这时被找出的弧就构成了可改进路 P。令改进量 a 是 $L(V_t)$，即 V_t 的第二个标号。

$$F_{ij}=\begin{cases}F_{ij}+a & (V_i,V_j)\in P+\\ F_{ij}-a & (V_i,V_j)\in P-\\ F_{ij} & (V_i,V_j)\notin P\end{cases}$$

去掉所有的标号，对新的可行流 $F'=\{F_{ij}\}$，重新进入标号过程。

三、求最大流的程序

```
program max_flow;
  const
    maxn          =  30;
  type
    nodetype      =  record { 可增广轨的顶点类型 }
                       l,p : integer; { 标号、检查标志 }
                     end;
    arctype       =  record { 网顶点类型 }
                       c,f : integer { 容量、流量 }
                     end;
    gtype         =  array [0..maxn,0..maxn] of arctype; { 网类型 }
    ltype         =  array [0..maxn] of nodetype;         { 可增广轨类型 }
  var
    lt            :  ltype; { 可增广轨 }
    g             :  gtype; { 网 }
    n,s,t         :  integer; { 顶点数、源点、汇点 }
    f             :  text;      { 文件变量 }
  procedure read_graph;
    var
      str : string;
      i,j : integer;
    begin
      write('Graph file = '); { 输入文件名，并与文件变量连接 }
        readln(str);
        assign(f,str);
        reset(f);
        readln(f,n); { 读入顶点数 }
        fillchar(g,sizeof(g),0);         { 网络初始化 }
        fillchar(lt,sizeof(lt),0);       { 增广矩阵初始化 }
        for i : =1 to n do               { 读入网矩阵 }
          for j : =1 to n do read(f,g[i,j].c);
        close(f);
```

```
  end;
function check : integer; { 返回一个已标号而未检查的顶点序号 }
  var
    i : integer;
  begin
    i : =1;
    while (i<=n) and not ((lt[i].l< >0) and (lt[i].p=0)) do inc(i);
    if i>n then check : =0
           else check : =i
  end;
function ford(var a : integer) : boolean;
{ 若 Vs 至 Vt 无增广轨返回 TRUE,否则返回 FALSE 和可增广轨的改进量 a }
  var
    i,j,m,x : integer;
  begin
    ford : =true; { 设无增广轨 }
    fillchar(lt,sizeof(lt),0);  { 增广轨撤消 }
    lt[s].l : =s;               { 从 Vs 开始寻找增广轨 }
    repeat
      i : =check;               { 寻找一个已标号而未检查的顶点序号 i }
      if i=0 then exit;  { 若该顶点不存在,则退出过程,返回 TRUE }
      for j : =1 to n do
        if (lt[j].l=0) and ((g[i,j].c<>0) or (g[j,i].c<>0))
         { 寻找所有与 i 连接、且未标号的顶点 j,弧<i,j>置前向或后向弧标志 }
          then begin
                 if (g[i,j].f<g[i,j].c) then lt[j].l : =i;
                 if (g[j,i].f>0) then lt[j].l : =-i
               end;
         lt[i].p : =1 { 顶点 i 的所有相关弧分析完,置顶点 i 检查标志 }
    until (lt[t].l<>0); { 循环,直至 t 顶点标号为止 }
    m : =t;a : =maxint; { 从 t 顶点倒推,改进量初始化 }
    repeat { 求 a=min[所有前向弧的 C(i,j)-F(i,j),所有后向弧的 F(i,j)]}
      j : =m;m : =abs(lt[j].l);
      if lt[j].l<0 then x : =g[j,m].f-0;
      if lt[j].l>0 then x : =g[m,j].c-g[m,j].f;
      if a>x then a : =x
    until m=s; { 直至倒推至 s 顶点为止 }
    ford : =false { 返回可增广轨存在标志 FALSE }
  end;
procedure fulkerson(a : integer);
  var
    m,j : integer;
  begin
    m : =t; { 从 t 顶点出发,沿增广轨修正流量 }
    repeat
      j : =m;m : =abs(lt[j].l);
      if lt[j].l<0 then g[j,m].f : =g[j,m].f-a;
      if lt[j].l>0 then g[m,j].f : =g[m,j].f+a
```

```
    until m=s
  end;
procedure answer; { 打印所有弧的流量 }
  var
    i,j : Integer;
  begin
    for i : =1 to n do
      for j : =1 to n do
        if g[i,j].f<>0 then writeln(i,'-----',j,' ',g[i,j].f)
  end;
procedure proceed;
  var
    del : integer;
    success : boolean;
  begin
    s : =1;t : =n; { 设置源点和汇点序号 }
    repeat
      success : =ford(del);          { 寻找可增广轨和可改进量 del }
      if success then answer   { 若可增广轨不存在,打印最大流 }
              else fulkerson(del)         { 否则沿增广轨修正流量 }
    until success
  end;
begin
  read_graph; { 输入网}
  proceed      { 求最大流 }
end.
```

6.2 求容量有上下界的网络的最大流和最小流

6.2.1 求容量有上下界的网络的最大流

一、问题及其求法

6.1 节讨论了网络的最大流问题。这种网络的每一条弧 e 对应了一个弧容量 $C(e)\geqslant 0$。现在,我们将网络结构再修改一下,每条弧 e 对应两个数字 $B(e)$和 $C(e)$,分别表示弧容量的上界和下界,那么如何求满足下列条件的最大流:

1. $B(e)\leqslant F(e)\leqslant C(e)$
2. $\sum\limits_{e\in\beta(V)} F(e)-\sum\limits_{e\in\alpha(V)} F(e)=0 \qquad (V\neq s,t)$

其中:$\alpha(V)$——以 V 为尾的弧集;

$\beta(V)$——以 V 为头的弧集。



图 6-4

显然,6.1 节例举的网络结构是有上下界的网络的一种特例,即 $B(e)=0$。当 $B(e)>0$ 时,这种有上下界的网络就不一定存在流了。例如,图 6-4 所示的网络,弧上的第 1 个数字为 $B(e)$的值,第 2 个数字为 $C(e)$的值。由

于 $0\leqslant F_{ta}\leqslant 1, 2\leqslant F_{at}\leqslant 3$，对于顶点 a 来讲，$F_{at}-F_{sa}\neq 0$，所以该网络不存在流。

那么如何判断一个有上下界的网络 N 有可行流呢？我们很容易想到一种思路——将容量有上下界的 $\bar{N}$ 网转换成 6.1 节所述的“每条弧仅对应一个容量 $C(e)\geqslant 0$”的附加网 $\bar{N}$，然后用标号法对其求最大流，根据求解结果判断 N 网是否有可行流 F。具体的转换方法如下：

1. 新增加两个顶点 $\bar{s}$ 和 $\bar{t}$，$\bar{s}$ 称为附加源，$\bar{t}$ 称为附加汇；

2. 对原网络 N 的每个顶点 U，加一条新弧 $e=U\bar{t}$，这条弧的容量为所有以 U 为尾的弧的容量下限之和，即 $\bar{C}(e)=\sum_{e\in\alpha(U)}B(e)$；

3. 对原网络 N 的每个顶点 U，加一条新弧 $e=\bar{s}U$，这条弧的容量为所有以 U 为头的弧的容量下限之和，即 $\bar{C}(e)=\sum_{e\in\beta(U)}B(e)$；

4. 原网络 N 的每条弧在 $\bar{C}$ 仍保留，弧容量 $\bar{C}(e)$ 修正为 $C(e)-B(e)$；

5. 再添两条新弧 $e=st, e'=ts$。e 的容量 $\bar{C}(e)=\infty$，e' 的容量 $\bar{C}(e')=\infty$。

用 6.1 节所述的办法求附加网 $\bar{N}$ 的最大流，若结果能使流出 $\bar{s}$ 的一切弧皆满载（即 $\bar{s}$ 出发的所有弧 e，$\bar{C}(e)=\bar{F}(e)$），则 N 网有可行流 $F(e)=\bar{F}(e)+B(e)$。否则 N 网无可行流。

若 $\bar{N}$ 网的最大流满足上述条件，我们则可以再使用 6.1 节所述的标号法，将可行流 F 放大，最终求出 N 网的最大流。

例如，图 6-5 为一个容量有上下界的网络 N；弧上界的数第 1 个是 $B(e)$，第 2 个是 $C(e)$。

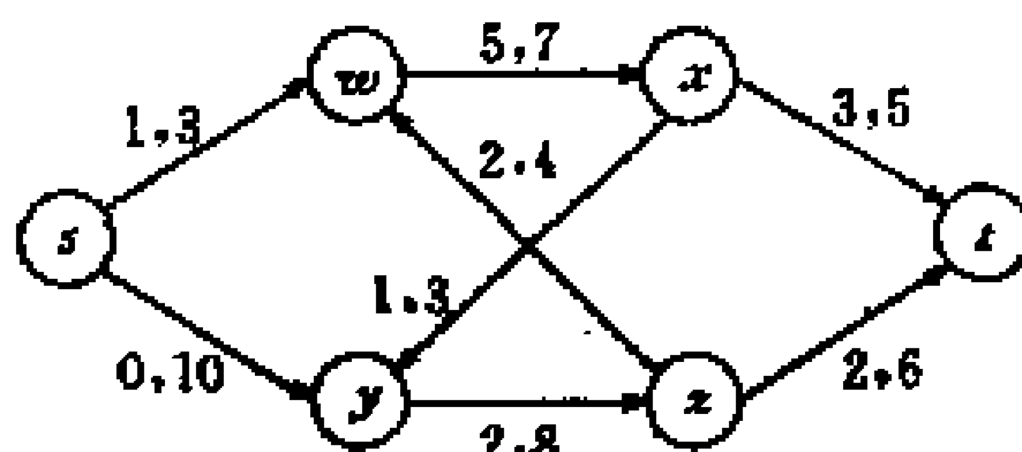


图 6-5

作附加网络 $\bar{N}$，用标号法求 $\bar{N}$ 网上的最大流 $\bar{F}$（见图 6-6）。

弧上的第 1 个数为 $\bar{C}(e)$，第 2 个数为 $\bar{F}(e)$。由于流出 $\bar{s}$ 的弧皆满载，可以得出图 6-5 的网 N 中有可行流 $F(e)=\bar{F}(e)+B(e)$。图 6-7 将 $\bar{F}$ 化成 N 网的可行流，弧上的 3 个数依次为 $B(e), C(e), F(e)$。

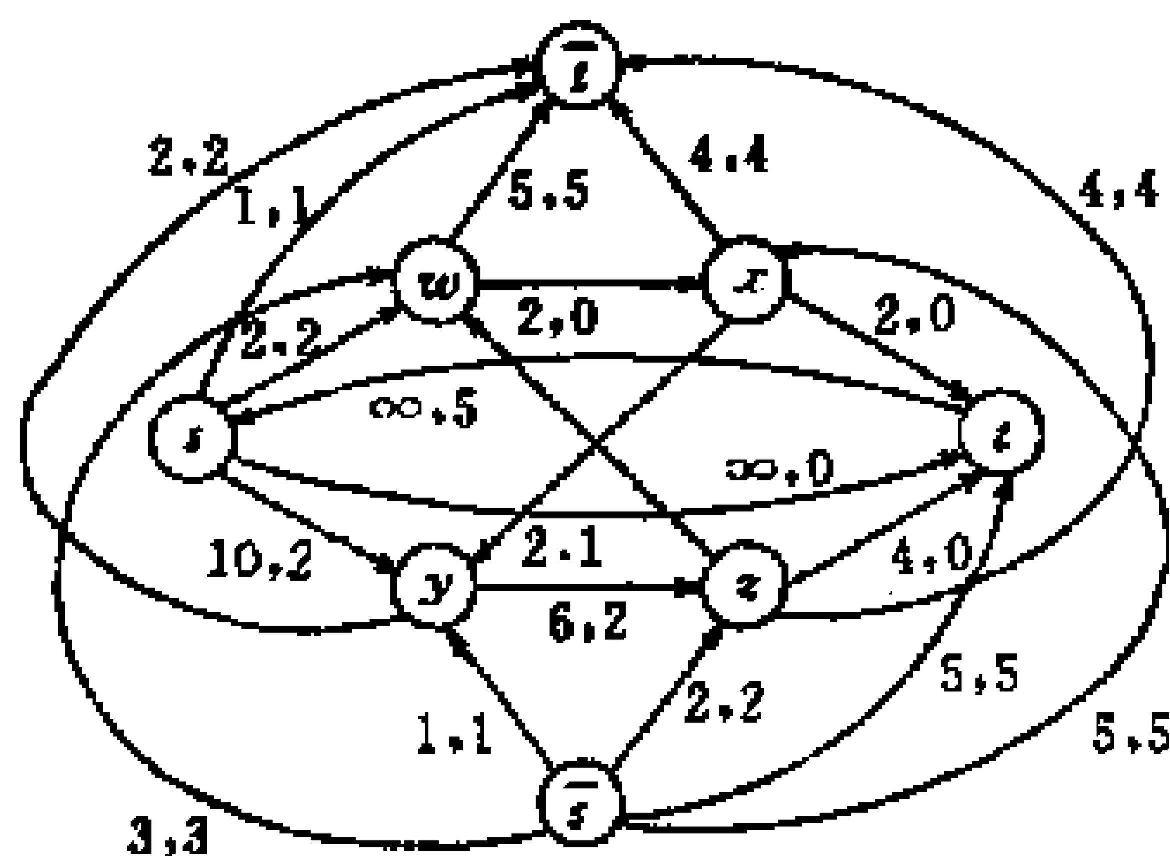


图 6-6

用 6.1 节所述的标号法将图 6-7 中的可行流放大，得最大流如图 6-8。

最大流的流量 $F=10$。

二、求容量有上下界的网络的最大流的程序

```
program max_flow_for_up_and_down;
{求容量有上下界的网络的最大流}
  const
```

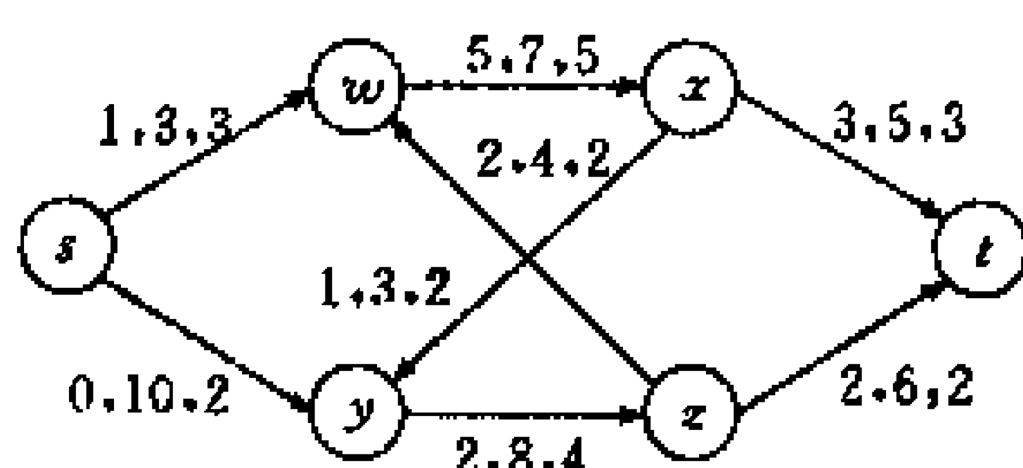


图 6-7

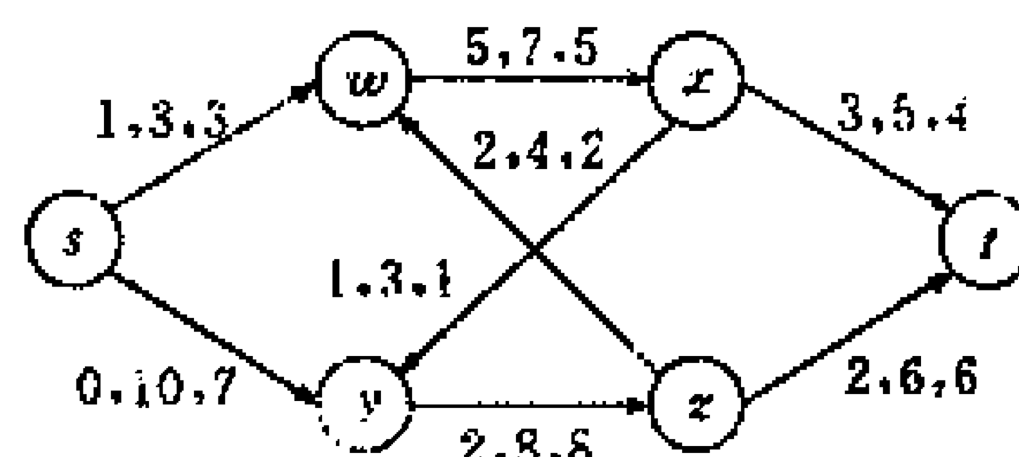


图 6-8

```
    maxn            =   30;
  type
    nodetype        =   record              { 增广轨的顶点类型 : }
                          l,p : integer;    { 标号、检查标志 }
                        end;
arctype             =   record              { 网的顶点类型: 容量上、下量、流量 }
                          b,c,f : integer
                        end;
    gtype           =   array [1..maxn,1..maxn] of arctype;   { 网类型 }
    ltype           =   array [1..maxn] of nodetype;          { 增广轨类型 }
  var
    lt              :   ltype;              { 增广轨 }
    g,gl            :   gtype;              { 网 }
    n,s,t           :   integer;            { 顶点数,源点,汇点 }
    f               :   text;               { 文件变量 }
  procedure read_graph;
    var
      str : string;
      i,j : integer;
    begin
      write('Graph file = ');  { 读入文件名并与文件变量连接 }
      readln(str);
      assign(f,str);
      reset(f);
      readln(f,n);                        { 读入顶点数 }
      fillchar(g,sizeof(g),0);      { 网络和可增广轨初始化 }
      fillchar(lt,sizeof(lt),0);  { 读入初始网 }
      for i := 2 to n+1 do
        for j := 2 to n+1 do read(f,g[i,j].b,g[i,j].c);
      close(f);
    end;
  function check : integer;  { 返回一个已标号而未检查的顶点序号 i }
    var
      i : integer;
    begin
      i := s;
      while (i<=n) and not ((lt[i].l<>0) and (lt[i].p=0)) do inc(i);
```

```
    if i>n then check : =0
          else check : =i
  end;
function ford(var a : integer) : boolean;
{ 若增广轨不存在返回 TRUE,否则返回 FALSE 和可改量 a }
  var
    i,j,m,x : integer;
  begin
    ford : =true; { 置增广轨不存在标志 }
    fillchar(lt,sizeof(lt),0); {增广轨撤空 }
    lt[s].l : =1; { 从 s 顶点开始标志增广轨 }
    repeat
      i : =check;
      { 搜索一个已标号而未检查的顶点序号,若该顶点不存在返回 true }
      if i=0 then exit;
      for j : =s to n do
        if (lt[j].l=0) and ((g[i,j].c<>0) or (g[j,i].c<>0))
        {搜索所有与 i 相连的未标号顶点 j,置<i,j>前向弧或后向弧标志 }
          then begin
                 if (g[i,j].f<g[i,j].c) then lt[j].l : =i;
                 if (g[j,i].f>g[j,i].b) then lt[j].l : =-i
               end;
      lt[i].p : =1                    { i 顶点已检查标志 }
  until (lt[t].l<>0);                 { 直至 t 顶点已标号为止 }
  m : =t;a : =maxint;                 { 从 t 顶点倒推,求可改进量 a }
  repeat
        j : =m;m : =abs(lt[j].l);
        if lt[j].l<0 then x : =g[j,m].f-g[j,m].b;
        if lt[j].l>0 then x : =g[m,j].c-g[m,j].f;
        if a>x then a : =x
      until m=s;
      ford : =false { 返回可增广轨存在标志 }
  end;
procedure fulkerson(a : integer); { 修正增广轨上的流量 }
  var
    m,j : integer;
  begin
    m : =t; { 从 t 顶点倒推 }
    repeat
      j : =m;m : =abs(lt[j].l);
      if lt[j].l<0 then g[j,m].f : =g[j,m].f-a;
      if lt[j].l>0 then g[m,j].f : =g[m,j].f+a
    until m=s { 直至倒推至 s 顶点为止 }
  end;
procedure answer; { 打印各弧上的最大流量 }
  var
    i,j : Integer;
  begin
```

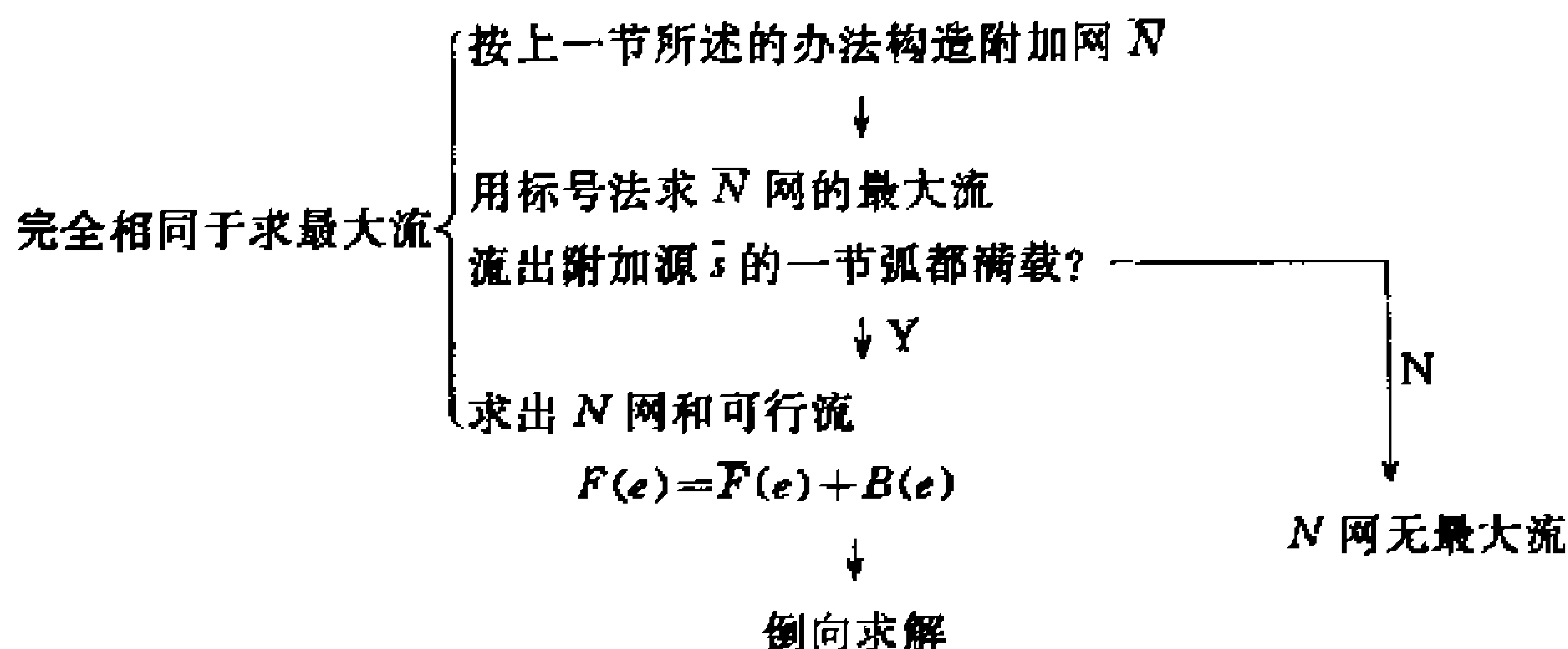
```
    for i : =2 to n do
      for j : =2 to n do
        if g[i,j].f<>0 then writeln(i-1,'————',j-1,' ',g[i,j].f)
  end;
procedure proceed;
  var
    i,j,x,del : integer;
    success : boolean;
  begin
    s : =1;inc(n,2);t : =n; { 增加一个附加源 s=1 和一个附加汇 t=n+2 }
    g1 : =g;fillchar(g,sizeof(g),0); { 暂存初始网,附加网初始化 }
    for i : =2 to n-1 do
      for j : =2 to n-1 do
      {
      {                           ┌加一条弧边 si = e,c'(e) = Σ(e∈β(i)) b(e)      b̄'(e) = 0 }
      { 对初始网的第一个顶点 i ┤                                                        }
      {                           │                                                        }
      {                           └加一条弧边 it = e,c'(e) = Σb(e)               b̄'(e) = 0 }
        begin
          g[s,i].c : =g[s,i].c+g1[j,i].b;
          g[i,t].c : =g[i,t].c+g1[i,j].b
        end;
    for i : =2 to n-1 do { 原网络的边保留,c'(e)=c(e)-b(e),b'(e)=0 }
      for j : =2 to n-1 do g[i,j].c : =g1[i,j].c-g1[i,j].b;
    g[2,n-1].c : =maxint;g[n-1,2].c : =maxint;
    { 增两条边 e1=st,e2=ts,c̄(e1)=c̄(e2)=∞,b̄(e1)=b̄(e2)=0 }
    repeat
      success : =ford(del); { 寻找新网中的可增广轨和改进量 del }
      if not success then fulkerson(del) { 若可增广轨存在,则修正流量 }
    until success; { 直至无增广轨存在、最大流求出 }
    for i : =2 to n-1 do { 求 f(e)=f̄(e)+b(e),成为初始网的一个可行流 }
      for j : =2 to n-1 do
        begin
        g1[i,j].f : =g[i,j].f+g1[i,j].b;
      end;
    s : =2;n : =n-1;t : =n;
    { 顶点 2 作为源点,顶点 n-1 为汇点,即恢复初始网的源点、汇点 }
    g : =g1; { 恢复初始网 }
    g[s,t].f : =0;g[t,s].f : =0;   { 撤去两条边(s,t),(t,s)两条边的流量 }
    repeat                          { 将原网的可行流放大,得最大流 }
      success : =ford(del);
      if success then answer
                 else fulkerson(del)
```

```
    until success
  end;
begin
  read_graph;  { 输入容量有上下界的网络 }
  proceed      { 计算并输出最大流 }
end.
```

6.2.2 求容量有上下界的网络的最小流

一、问题及其算法

对于一个容量有上下界的网络 N，我们已经有了对其求最大流的算法。因此，我们不难得出求 N 网的最小流的算法：



以 t 为源点，s 为汇点，用标号法放大可行流 F，最终求出的最大流即为从 s 到 t 的最小流。

二、求容量有上下界的网络的最小流的程序

```
program min_flow_for_up_and_down;
{求容量有上下界的网络的最小流}
  const
    maxn        =  30;
  type
    nodetype    =  record                          { 增广轨的顶点类型 }
                     l,p : integer;                { 标号,检查标志 }
                   end;
    arctype     =  record                          { 网的顶点类型 }
                     b,c,f : integer               { 容量上下界,流量 }
                   end;
    gtype       =  array [1..maxn,1..maxn] of arctype;   { 网类型 }
    ltype       =  array [1..maxn] of nodetype;          { 增广轨类型 }
  var
    lt          :  ltype;                          { 增广轨 }
    g,gl        :  gtype;                          { 网 }
```

```
  n,s,t          : integer;                    { 顶点数,源点,汇点 }
  f              : text;                       { 文件变量 }
procedure read_graph;
  var
    str : string;
    i,j : integer;
  begin
    write('Graph file = ');                    { 读入文件名并与文件变量连接 }
    readln(str);
    assign(f,str);
    reset(f);
    readln(f,n);                               { 读入顶点数 }
    fillchar(g,sizeof(g),0);                   { 网络和可增广轨初始化 }
    fillchar(lt,sizeof(lt),0);                 { 读入初始网 }
    for i : =2 to n+1 do
      for j : =2 to n+1 do read(f,g[i,j].b,g[i,j].c);
    close(f);
  end;
function check : integer;                      { 返回一个已标号而未检查的顶点序号 i }
  var
    i : integer;
  begin
    i : =s;
    while (i<=n) and not ((lt[i].l<>0) and (lt[i].p=0)) do inc(i);
    if i>n then : integer) : boolean;
{ 若增广轨不存在返回 TRUE,否则返回 FALSE 和可改量 a }
  var
    i,j,m,x : integer;
  begin
    ford : =true; { 置增广轨不存在标志 }
    fillchar(lt,sizeof(lt),0); {增广轨撤空 }
    lt[s].l : =1; { 从 s 顶点开始标志增广轨 }
    repeat
      i : =check;
      { 搜索一个已标号而未检查的顶点序号,若该顶点不存在返回 true }
      if i=0 then exit;
      for j : =s to n do
        if (lt[j].l=0) and ((g[i,j].c<>0) or (g[j,i].c<>0))
        {搜索所有与 i 相连的未标号顶点 j,置<i,j>前向弧或后向弧标志 }
          then begin
                  if (g[i,j].f<g[i,j].c) then lt[j].l : =i;
                  if (g[j,i].f>g[j,i].b) then lt[j].l : =-i
               end;
      lt[i].p : =1             { i 顶点已检查标志 }
    until (lt[t].l<>0);        { 直至 t 顶点已标号为止 }
    m : =t;a : =maxint;        { 从 t 顶点倒推,求可改进量 a }
    repeat
      j : =m;m : =abs(lt[j].l);
      if lt[j].l<0 then x : =g[j,m].f-g[j,m].b;
```

```
      if lt[j].l>0 then x : =g[m,j].c-g[m,j].f;
      if a>x then a : =x
    until m=s;
    ford : =false { 返回可增广轨存在标志 }
  end;
procedure fulkerson(a : integer); { 修正增广轨上的流量 }
  var
    m,j : integer;
  begin
    m : =t; { 从 t 顶点倒推 }
    repeat
      j : =m;m : =abs(lt[j].l);
      if lt[j].l<0 then g[j,m].f : =g[j,m].f-a;
      if lt[j].l>0 then g[m,j].f : =g[m,j].f+a
    until m=s { 倒推直至 s 顶点为止 }
  end;
procedure answer; { 打印各弧上的最大流量 }
  var
    i,j : Integer;
  begin
    for i : =2 to n do
    for j : =2 to n do
      if g[i,j].f<>0 then writeln(i-1,'————',j-1,' ',g[i,j].f)
  end;
procedure proceed;
  var
    i,j,x,del : integer;
    success : boolean;
  begin
    s : =1;inc(n,2);t : =n; { 增加一个附加源 s=1 和一个附加汇 t=n+2 }
    g1 : =g;fillchar(g,sizeof(g),0); { 暂存初始网,附加网初始化 }
    for i : =2 to n-1 do
      for j : =2 to n-1 do
      {                          ⎧加一条弧边 si = e,c'(e) = Σ(e∈β(i)) b(e)      b'(e) = 0 }
      {                          ⎪
      { 对初始网的第一个顶点 i ⎨                                                    }
      {                          ⎪                                                    }
      {                          ⎩加一条弧边 it = e,c'(e) = Σ(e∈α(i)) b(e)      b'(e) = 0 }
        begin
          g[s,i].c : =g[s,i].c+g1[j,i].b;
          g[i,t].c : =g[i,t].c+g1[i,j].b
        end;
    for i : =2 to n-1 do { 原网络的边保留,c'(e)=c(e)-b(e),b'(e)=0 }
      for j : =2 to n-1 do g[i,j].c : =g1[i,j].c-g1[i,j].b;
    g[2,n-1].c : =maxint;g[n-1,2].c : =maxint;
```

```
  { 增两条边 e1=st,e2=ts,c̄(e1)=c̄(e2)=∞,b̄(e1)=b̄(e2)=0 }
  repeat
    success : =ford(del); { 寻找新网中的可增广轨和改进量 del }
    if not success then fulkerson(del) { 若增广轨存在,则修正流量 }
  until success; { 直至无增广轨存在、最大流求出 }
  for i : =2 to n-1 do { 求 f(e)=f̄(e)+b(e),成为初始网的一个可行流 }
    for j : =2 to n-1 do
      begin
      g1[i,j].f : =g[i,j].f+g1[i,j].b;
    end;
  t : =2;n : =n-1;s : =n;
  { 顶点 n-1 作为源点,顶点 2 为汇点,相对初始网来说 }
  { 求 t 至 s 的最大流,即求 s 至 t 的最小流 }
  g : =g1; { 恢复初始网 }
  g[s,t].f : =0;g[t,s].f : =0;  { 撤去两条边(s,t),(t,s)两条边的流量 }
  repeat                        { 将原网的可行流放大,得最大流 }
    success : =ford(del);
    if success then answer
             else fulkerson(del)
  until success
 end;

begin
 read_graph;  { 输入容量有上下界的网络 }
 proceed      { 计算并输出最小流 }
end.
```

6.3 最小费用最大流问题

6.1 节讨论了寻找网络中的最大流问题。在实际生活中,涉及“流”的问题时,人们考虑的还不只是流量,而且还有“费用”的因素。例如,图 6-9 是一个公路网,V_s 是仓库乙所在地,即物资的起点,V_t 是甲地,即物资的终点。另外,每一条弧旁都写了两个数字。第 1 个数字表示某一时间里通过这段公路的最多吨数,第 2 个数字表示每吨物资通过该公路的费用。现在的问题是怎样安排运输才能既使得从起点 V_s 运到终点 V_t 的物资最多,又使得总的运输费用最少?

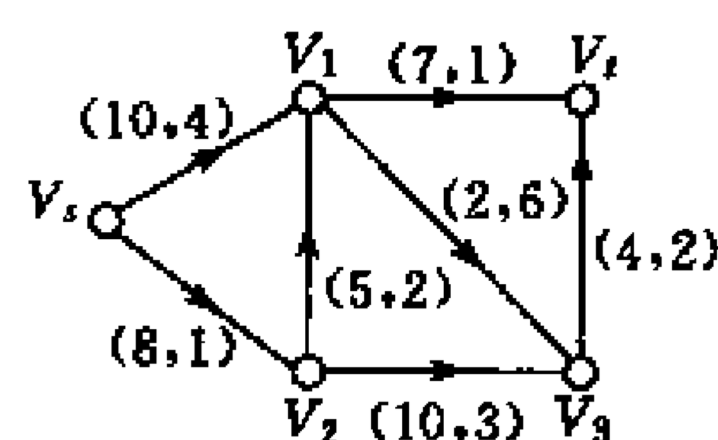


图 6-9 (弧旁的数字为(容量,单位流量费用))

显然图 6-9 是一个网络。这个网络的每一条弧(V_i,V_j)除给定的容量 C_{ij}外,还给了一个单位流量费用 $B_{ij}\geqslant 0$。上述问题的数学模型即要求一个最大流 F,使流的总输送费用

$$B(F)=\sum_{(V_i,V_j)\in A} B_{ij}*F_{ij} \qquad \text{取极小值}$$

这即所谓的最小费用最大流问题。

一、最小费用最大流的算法

从 6.1 节可知，寻找最大流的方法是从某个可行流出发，找到关于这个流的一条可改进路 P，沿着 P 调整 F，对新的可行流试图寻找关于它的可改进路，如此反复直至求得最大流。现在要寻求最小费用的最大流，我们首先考察一下，当沿着一条关于可行流 F 的可改进路 P，以 $a=1$ 调整 F，得到新的可行流 F'（显然$V(F')=V(F')+1$），$B(F')$ 比 $B(F)$ 增加多少？

不难看出：

$$B(F')-B(F)=\left[\sum_{P+} B_{ij}*(F'_{ij}-F_{ij})-\sum_{P-} B_{ij}*(F'_{ij}-F_{ij})\right]$$
$$=\sum_{P+} B_{ij}-\sum_{P-} B_{ij}$$

我们把$\sum_{P+} B_{ij}-\sum_{P-} B_{ij}$ 称为这条可行路 P 的"费用"。

显然，若 F 是流量为 $V(F)$的所有可行流中费用最小者，而 P 是关于 F 的所有可改进路中费用最小的可改进路，那么沿 P 去调整 F，得到的可行流 F'，即流量为 $V(F')$的所有可改进流中的最小费用流。这样，当 F 是最大流时，它也就是我们所要求的最小费用最大流了。

注意，由于 $B_{ij}\geqslant 0$，所以 $F=0$ 必是流量为 0 的最小费用流。这样，总可以从 $F=0$ 开始。一般，设已知 F 是流量 $V(F)$的最小费用流，余下的问题是如何去寻求关于 F 的最小费用可改进路。为此我们构造一个赋权有向图 $W(F)$，它的顶点是原网络 D 的顶点，而把 D 中的每一条弧(V_i,V_j)变成两个方向相反的弧(V_i,V_j)和(V_j,V_i)。定义 $W(F)$中弧的权 W_{ij}为：

$$W_{ij}=\begin{cases}B_{ij} & \text{若 } F_{ij}<C_{ij}\\ +\infty & \text{若 } F_{ij}=C_{ij}\end{cases}$$

$$W_{ji}=\begin{cases}-B_{ij} & \text{若 } F_{ij}>0\\ +\infty & \text{若 } F_{ij}=0\end{cases}$$

（长度为 ∞ 的可以从 $W(F)$ 中略去）

于是在网络中寻求关于 F 的最小费用可改进路，就等价于在赋权有向图 $W(F)$中，寻求从 V_s 到 V_t 的最短路。因此有如下算法：

开始取 $F(0)=0$，一般若在第 $k-1$ 步得到最小费用流 $F(k-1)$，则构造赋权有向图 $W(F(k-1))$，在 $W(F(k-1))$中，寻求从 V_s 到 V_t 的最短路。若不存在最短路（即最短路权是$+\infty$），则 $F(k-1)$即最小费用最大流；若存在最短路，则在原网络 D 中得到相应的可改进路 P，在可改进路 P 上对 $F(k-1)$进行调整：

$$a=\min\left[\min_{P+}(C_{ij}-F_{ij}(k-1)),\ \min_{P-}F_{ij}(k-1)\right]$$

$$F_{ij}(k)=\begin{cases}F_{ij}(k-1)+a & (V_i,V_j)\in P+\\ F_{ij}(k-1)-a & (V_i,V_j)\in P-\\ F_{ij}(k-1) & (V_i,V_j)\notin P\end{cases}$$

得到新的可行流 $F(k)$，再对 $F(k)$ 重复上述步骤。

例如，求图 6-9 的最小费用最大流。弧旁数字为 (C_{ij},B_{ij})。

(1) 取 $F(0)=0$ 为初始可行流；

(2) 构造赋权有向图 $W(F(0))$，并求出从 V_s 到 V_t 的最短路 (V_s,V_2,V_1,V_t)，如图 6-10(a)(双箭头即最短路)；

(3) 在原网络 d 中，与这条最短路相应的可改进流为 $P=(V_s,V_2,V_1,V_t)$；

(4) 在 P 上进行调整，$a=5$，得 $F(1)$(图 6-10(b))。按照上述算法依次得 $F(1)$，$F(2)$，$F(3)$，$F(4)$ 的流量依次为 5，7，10，11(见图 6-10(b)，(d)，(f)，(h))；构造相应的赋权有向图为 $W(F(1))$，$W(F(2))$，$W(F(3))$，$W(F(4))$(见图 6-10(c)，(e)，(g)，(i))。

注意到 $W(F(4))$ 中已不存在从 V_s 到 V_t 的最短路，所以 $F(4)$ 为最小费用最大流。

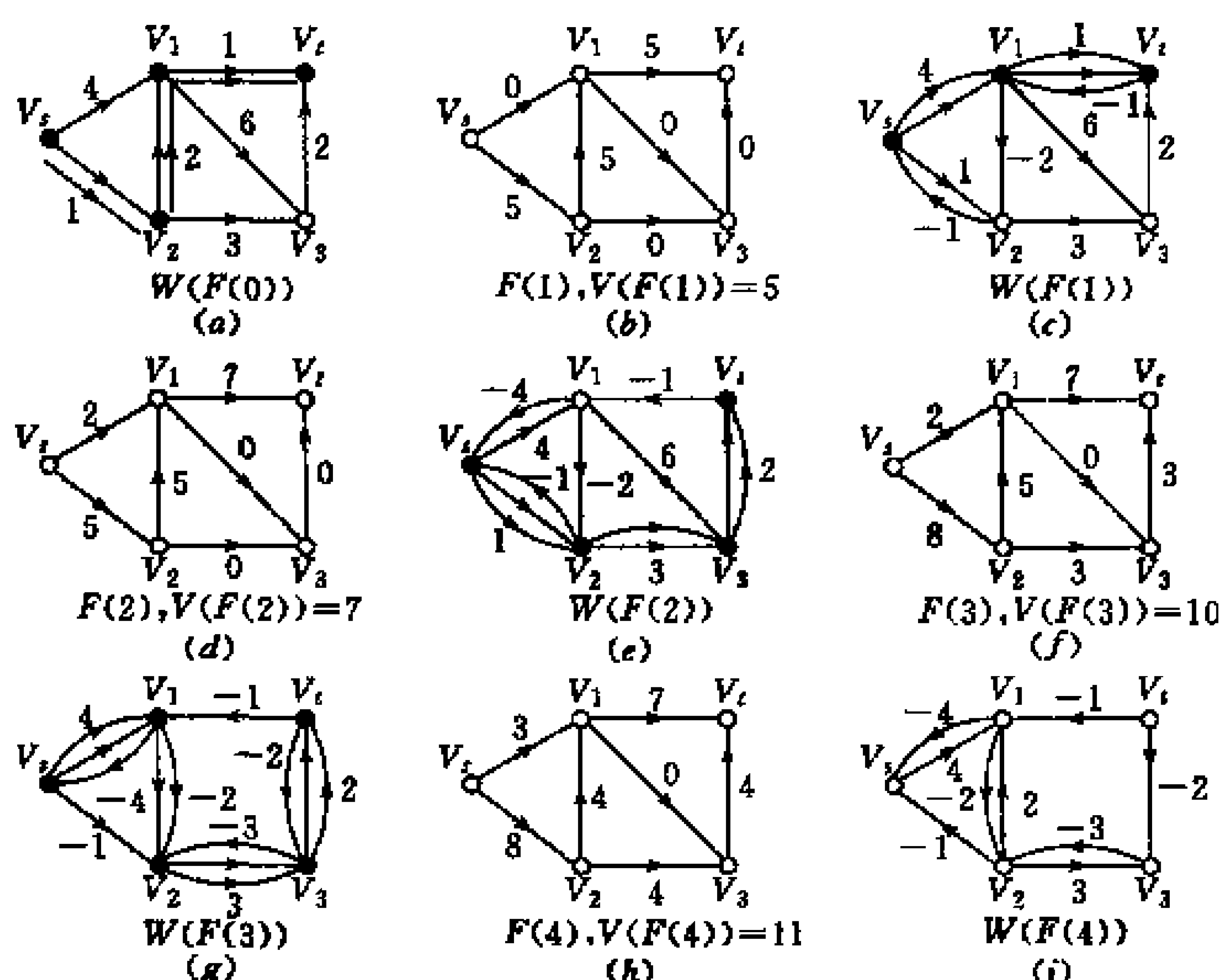


图 6-10

二、最小费用最大流的程序

```
program max_flow;
type arctype=record{弧结点类型：}
            max : integer;{容量}
            act : integer;{流量}
            end;
   settype=set of byte;
var
  n1,node : array [1..20] of integer;   {辅助变量,最佳路径序列}
  arc : array[1..20,1..20]of arctype;   {网络}
  w,b : array[1..20,1..20]of integer;   {最小费用和单位费用}
  success : boolean;                    {成功标志}
```

```
  start,ends,a,n,min : integer;
  {源点、汇点、改进量、结点个数、最佳代价}
procedure getting_path(s,tot : integer;m : settype);
  {求从 start 至 ends 的最佳路径,若无最佳路径,则 min=maxint}
  {s——待扩展结点,tot——当前路径代价,m——当前最佳路径结点集合}
  var
    i : integer;
  begin
    if s=ends
      then begin
             if tot<min
               then begin
                      min : =tot; node : =n1
                    end
           end
      else  begin
             for i : =1 to n do
               if (w[s,i]<>maxint) and not (i in m)
                 then begin
                        if arc[s,i].act<arc[s,i].max then n1[i] : =s;
                        if arc[i,s].act>0 then n1[i] : =-s;
                        if i=ends then n1[i] : =s;
                        getting_path(i,tot+w[s,i],m+[i])
                      end
           end
  end;
function ford(var a : integer) : boolean;
{ 若 w 图无从 start 至 ends 的最佳路径,则返回 true;否则进行标号过程并返回 false}
{ 和改进量 a}
  var
    i,j,m,s : integer;
  begin
    ford : =false;
    for i : =1 to n do {w 图初始化}
      for j : =1 to n do w[i,j] : =maxint;
    for i : =1 to n do {根据当前可行流量,给 w 图赋权}
      for j : =1 to n do
        with arc[i,j] do
          if max>0 then
            begin
              if act<max then w[i,j] : =b[i,j];
              if act=max then w[i,j] : =maxint;
              if act>0 then w[j,i] : =-b[i,j];
              if act=0 then w[j,i] : =maxint;
            end;{with}
    n1[start] : =start;{初始结点进入增广链}
    min : =maxint;     {最佳路径代价初始化}
    getting_path(start,0,[start]); {求最佳路径}
    if min=maxint      {若无最佳路径,则成功退出}
```

```
      then begin
            ford : =true;exit
          end;
    m : =ends;a : =maxint;    {从 vn 开始}
    repeat              {对最佳路径表示的增广链求改进量 a}
      j : =m;m : =abs(node[j]);
      if node[j]<0 then s : =arc[j,m].max;
      if node[j]>0 then s : =arc[m,j].max-arc[m,j].act;
      if a>s then a : =s;
    until m=start;
  end;{ford}
procedure fulkerson(a : integer);{对增广链进行调整}
  var
    i,m,j : integer;
  begin
    m : =ends;
    repeat
      j : =m;m : =abs(node[j]);
      if node[j]<0 then arc[j,m].act : =arc[j,m].act-a;{改进后向弧}
      if node[j]>0 then arc[m,j].act : =arc[m,j].act+a;{改进前向弧}
    until m=start;
  end;{fulkerson}
procedure answer; {打印最小费用最大流和总输送费用}
  var
    i,j,tot : integer;
  begin
    tot : =0;
    for i : =1 to n do
      for j : =1 to n do
        if arc[i,j].max<>0
          then begin
                 writeln (i,'————',j,'的流量和费用',arc[i,j].act,' ',
                        b[i,j],' ',b[i,j] * arc[i,j].act);
                 tot : =tot+b[i,j] * arc[i,j].act
               end;
    writeln('Tot = ',tot)
  end;{answer}
procedure init; {数据输入}
  var
    s,t,i,m : integer;
  begin
    fillchar(arc,sizeof(arc),0);
    write('结点数 : ');readln(n);
    write('弧数 : ');readln(m);
    write('初始结点 : ');readln(start);
    write('终止结点 : ');readln(ends);
    for i : =1 to m do
      begin
```

```
      repeat
        write('第',i,'条弧的(始点)(尾点)');
        readln(s,t);
        until (s in [1..n])and (t in [1..n])and (s<>t);
      write('第',i,'条弧的(容量)和(费用)');readln(arc[s,t].max,b[s,t]);
      end;{for}
   end;{init}

begin
  init;                   {输入含上下界和费用的网络}
  repeat                  {重复标号、调整过程,直至求出最小费用最大流}
    success := ford(a);
    if not success then fulkerson(a);
  until (success) ;
  answer; {打印最小费用最大流}
end. {main}
```

三、求最小费用最大流的一个实例——最佳航空路线问题

1. 问题及其分析

你在加拿大航空公司组织的一次竞赛中获奖,奖品是一张免费机票,可在加拿大旅行,从最西的一个城市出发,单方向从西向东经若干城市到达最后一个城市(必须到达最东的城市),然后再单方向从东向西飞回起点(可途经若干城市)。除起点城市外,任何城市只能访问一次。起点城市被访问二次:出发一次,返回一次。除指定的航线外,不允许乘其它航线,也不允许使用其他交通工具。求解的问题是:给定城市表列及两城市之间的直通航线表列,请找出一条旅行航线,在满足上述限制条件下,使访问的城市尽可能多。

多个不同的输入数据组写在一个名为 C:\IOI\ITIN.DAT 的 ASCII 文件中,文件中每一数据组的格式说明如下:

- 数据组的第一行是 n 和 V;

  n 代表可以被访问的城市数,n 是正整数,$n<100$;

  V 代表下面要列出的直飞航线数,V 是正整数。

- 以下 n 行中每一行是一个城市名,可乘飞机访问这些城市。城市名出现的顺序是:从西向东。也就是说,设 i,j 代表城市表列中城市出现的顺序,当 $i>j$ 时,表示城 i 在城 j 的东边(这里保证不会有两个城市在同一经线上)。

  城市名是一个长度不超过 15 的字符串,串中的字符可以是字母或阿拉伯数字。

  例如:AGR34 或 BEL4

- 接下来的 V 行中,每行有两个城市名,中间用空格隔开,如下所示:

  City1 City2

  表示从 City1 到 City2 有一条直通航线,从 City1 到 City2 也有一条直通路线。

- 不同的输入数据组之间被空行隔开(参看例子),最后一个数据组之后没有空行。

下面的例子放在文件 C:\IOI\ITIN.DAT 中:

```
8 9
Vancouver
```

```
Yellowknife
Edmonton
Calgary
Winnipeg
Toronto
Montreal
Halifax
Vancouver     Edmonton
Vancouver     Calgary
Calgary       Winnipeg
Winnipeg      Toronto
Toronto       Halifax
Montreal      Halifax
Edmonton      Montreal
Edmonton      Yellowknife
Edmonton      Calgary
```

5 5

C_1

C_2

C_3

C_4

C_5

C_5 C_4

C_2 C_3

C_2 C_1

C_4 C_1

C_5 C_2

假定输入数据完全正确，不必对输入数据进行检查。

对于每一组输入数据，其结果都必须写在名为 C：\IOI\ITIN. SOL 的 ASCII 文件中，其格式为：

- 第 1 行是输入数据中给出的城市数；
- 第 2 行是你所建立的旅行路线中所访问的城市总数 M；
- 接下来的(M+1)行是你的旅行路线中的城市名，每行写一个城市名。首先是出发城市名，然后按访问顺序列出其它城市名。注意：最后一行（终点城市）的城市名必须是出发城市名。
- 如题目无解，输出数据格式为：

第 1 行是输入数据中给出的城市数；

第 2 行写："NO SOLUTION"。

上述例子的解如下所示：

```
ITIN. SOL
8
7
Vancouver
Edmonton
```

Montreal
Halifax
Toronto
Winnipeg
Calgary
Vanouver

5
NO SOLUTION
你的源程序文件名为：C：\IOI\DDD.PAS

上述题目可以用搜索算法，但问题是，一旦城市数或直通航线数多时便很难求出解。现在有了求最小费用最大流的算法基础，就可以迅速地求得这条最佳的旅行路线。

首先，按下述原则构造一个新的网络 n：

(1) 每个城市 i 拆成两个顶点 i 和 i'，并在两个顶点之间连接一条由 i 至 i' 的有向弧，弧的容量为 1，表示该城市最多只能访问一次。为了使该城市尽可能被访问，单位流量费用设为 0；

(2) 若城市 i 到城市 j 有直通航线($i<j$)，则在顶点 i' 与顶点 j 之间连接一条弧，方向由顶点 i' 至顶点 j。弧的容量为 1，表示这条航线最多只能通过一次。单位流量费用设 $j-i+1$，即城市 j 位于城市 i 的东面，中间相隔的城市数为 $j-i+1$；

(3) 顶点 1 与顶点 2 之间的弧容量改为 2，表示最西端的城市 1 被访问两次。顶点 n 与顶点 n' 的弧容量也改为 2，因为如果往返航线分作两条路线考虑的话，则最东端的城市 n 可以看作为被访问两次；

(4) 顶点 1 作为源点，顶点 n' 作为汇点(图 6-11)。

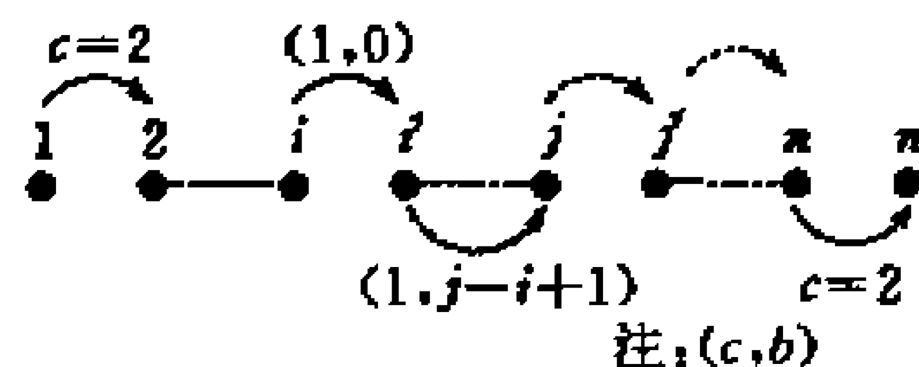


图 6-11

然后对上述网络 n 求最小费用最大流，其结果形成：

一条由顶点 1 至顶点 n、弧容量为 1 且互相连接的若干条弧组成的流，表示往程航线；另一条由顶点 n 至顶点 1、弧容量为 1 且互相连接的若干条弧组成的流，表示返程航线；这两条流上的[(顶点序号+1)/2]即为往返航线上被访问的城市序号。

显然最佳的旅行路线上经过的城市数$=2\times(n-1)-$流量费用。若可以经过所有城市的话，则城市数＝流量费用＋2。

2. 程序题解

```
program air_lines;
{航空问题}
```

```
const
  maxn          =  100;
type
  node          =  record                              { 网顶点类型 : }
                     b,c,f,w : shortint;               { 上界、单位费用、流量、权 }
                   end;
  gtype         =  array [1..maxn,1..maxn] of node; { 网类型 }
  pn            =  record                              {可增广轨顶点类型 : }
                     l,c   :   integer                 { 标号、最短路径长度 }
                   end;
  nametype      =  string[20]; { 城市名串 }
var
  f             :  text; { 文件变量 }
  n,s,t,v       :  integer; { 城市数、源点、汇点、航线数 }
  l             :  array [1..maxn] of pn; { 可增广轨 }
  pm            :  array [1..maxn,1..maxn] of boolean; { 邻接矩阵 }
  g             :  gtype; { 网 }
  nl            :  array [1..maxn] of nametype; { 城市名序列 }
function p(s : nametype) : integer; { 返回城市名 s 对应的序号 }
  var
    i : integer;
  begin
    for i : =1 to n do
      if nl[i]=s then
        begin
          p : =i;exit
        end
  end;
procedure init;
  var
    i,j : Integer;
    str,s1,s2 : string;
  begin
    write('File name = '); { 读入文件名并与文件变量连接 }
    readln(str);
    assign(f,str);
    reset(f); { 读准备 }
    readln(f,n,v); { 读入城市数和航线数 }
    for i : =1 to n do readln(f,nl[i]); { 读入城市名序列 }
    fillchar(pm,sizeof(pm),0);
    for i : =1 to v do { 依次读入每条直通航线 }
      begin
        readln(f,str);
        s1 : =copy(str,1,pos(' ',str)-1);
        s2 : =copy(str,pos(' ',str)+1,length(str)-pos(' ',str));
        pm[p(s1),p(s2)] : =true;
        pm[p(s2),p(s1)] : =true
      end
```

```
  end;
procedure make_graph; { 构造新图 N }
  var
    i,j : integer;
  begin
    fillchar(g,sizeof(g),$fe); { -2 }
    for i := 1 to n do
      begin
        with g[2*i-1,2*i] do { 顶点 2i-1 至顶点 2i 连一条弧，上界为 1，费用为 0 }
          begin
            b := 1;c := 0;f := 0
          end;
        for j := i+1 to n do
          if pm[i,j] then
{ 若城市 i 到城市 j 有航线(i<j)，则在顶点 2i 至顶点 2j-1 连一条弧，上界为 1}
            with g[2*i,2*j-1] do                    { 费用为 j-i-1 }
              begin
                b := 1;c := j-i-1;f := 0
              end;
      end;
    g[1,2].b := 2;g[2*n-1,2*n].b := 2; { 设弧(1,2)和弧(2n-1,2n)的上界为 2 }
    s := 1;t := 2*n { 源点为顶点 1，汇点为顶点 2n }
  end;
function shortest : boolean; { 若存在最短路径，返回 true，否则返回 false }
  var
    i,j : integer;
    brk : boolean;
  begin
    for i := s to t do l[i].c := maxint;
    l[s].l := 0;l[s].c := 0;
    repeat
      brk := true;
      for i := s to t do
        for j := s to t do
          if (g[i,j].w<>-2) and (l[i].c<>maxint) and (l[i].c+g[i,j].w<l[j].c)
            then begin
                   l[j].c := l[i].c+g[i,j].w;
                   if g[i,j].f<g[i,j].b then l[j].l := i;
                   if g[j,i].f>0 then l[j].l := -i;
                   brk := false
                 end
    until brk;
    if l[t].c=maxint then shortest := false
                     else shortest := true
  end;
function ford : boolean;
  var
    i,j : integer;
```

```
  begin
    for i:=s to t do
      for j:=s to t do g[i,j].w:=-2;
    for i:=s to t do { 根据当前可行流量,给 w 图赋权 }
      for j:=s to t do
        if g[i,j].f<>-2
          then begin
                 if g[i,j].f<g[i,j].b then g[i,j].w:=g[i,j].c;
                 if g[i,j].f>0 then g[j,i].w:=-g[i,j].c
               end;
    ford:=shortest { 在 w 图中寻找从顶点 s 至顶点 t 的最短路 }
  end;
procedure fulkerson;
  var
    i,j,delta:integer;
  begin
    delta:=maxint;
    i:=t; { 从顶点 t 开始,倒向计算调整量 delta }
    repeat
      if l[i].l>0 then j:=g[l[i].l,i].b-g[l[i].l,i].f;
      if l[i].l<0 then j:=g[i,-l[i].l].f;
      if j<delta then delta:=j;
      i:=abs(l[i].l)
    until i=s;
    i:=t; { 从 t 顶点开始,倒向调整增广轨上的弧流量 }
    repeat
      if l[i].l>0 then inc(g[l[i].l,i].f,delta);
      if l[i].l<0 then inc(g[i,-l[i].l].f,delta);
      i:=abs(l[i].l)
    until i=s
  end;
procedure proceed; { 求 N 网的最小费用最大流 }
  begin
    while ford do fulkerson
  end;
procedure answer;
  var
    i,j,k:integer;
  begin
    if g[1,2].f<>2 { 若未返回城市 1,则无解 }
      then begin
             writeln('No Answer');halt
           end;
    k:=0;
    writeln('Answer : ');
    i:=s; { 从城市 1 出发,打印往程航线 }
    while i<>t-1 do
      begin
```

```
        write(nl[(i+1) div 2],'--');
        j:=i+2;
        while (g[i+1,j].f<>1) do inc(j);
        g[i+1,j].f:=-g[i+1,j].f;i:=j;inc(k);
      end;
    writeln(nl[(i+1) div 2]); { 从城市 n 出发,打印返城航线 }
    while i<>s do
      begin
        write(nl[(i+1) div 2],'--');
        j:=i-1;
        while g[j,i].f<>1 do dec(j);
        g[j,i].f:=-g[j,i].f;i:=j-1;inc(k);
      end;
    writeln(nl[(i+1) div 2]);
    writeln('Have travelled ',k,' cities');
    writeln('OK!');
    readln
  end;

  begin
    init;             { 读入航空图 }
    make_graph;       { 构造 N 网 }
    proceed;          { 求 N 网的最小费用最大流 }
    answer            { 输出最佳航空路线 }
  end.
```

6.4 求容量有上下界的网络的最小费用最小流和应用实例

6.4.1 求容量有上下界的网络的最小费用最小流

一、问题及其求解法

现有一个网络 N,N 的每条弧 e 对应 3 个数字:$B(e)$和 $C(e)$分别表示弧容量的上界和下界;$A(e)$表示弧的单位流量费用。所谓求容量有上下界的网络的最小费用最小流,即求满足下列条件的最小流 F_0:

1. $B(e)\leqslant F(e)\leqslant C(e)$;

2. $\sum_{e\in\beta(U)} F(e) - \sum_{e\in\alpha(U)} F(e) = 0 \qquad U \neq s,t$;

3. 流的总运输费用 $B(F) = \sum A(e)\times F(e)$ 取极小值。

实际生活中,很多问题可转换为求最小费用最小流。例如下一节所述的餐厅问题,就是其中的一例。有了 6.2 节求容量有上下界的网络的最大流和 6.3 节求最小费用最小流的基础。我们不难得出解决这个问题的一种方法:

1. 按 6.2 节所述的办法,构造一个附加网 $\overline{N}$:

(1) 新增两个顶点 $\bar{s},\bar{t}$,其中 $\bar{s}$ 称作附加源,$\bar{t}$ 称作附加汇;

(2) 对原网络 N 的每个顶点加一条新弧 $e=U\bar{t}$,这条弧的容量为所有 $\alpha(U)$容量的下

限之和，即 $\bar{C}(e) = \sum_{e \in \alpha(U)} B(e)$；

(3) 对原网络 N 的每个顶点 U 加一条新弧 $e=sU$，这条弧的容量为所有以 U 为头的弧容量的下限之和，即 $\bar{C}(e) = \sum_{e \in \beta(U)} B(e)$；

(4) 原网络 N 的每条弧在 N 仍保留，弧容量 $C(e)$ 修正为 $C(e)-B(e)$；

(5) 再添两条新弧 $e=st, e=ts$。e 的容量 $\bar{C}(e)=\infty$，$\bar{e}$ 的容量 $\bar{C}(\bar{e})=\infty$。

2. 求附加网 N 的最大流。若结果能使流出 $\bar{s}$ 的一切弧皆满载（即 $\bar{s}$ 出发的所有弧 e，$\bar{C}(e)=\bar{F}(e)$），则 N 网有可行流 $F(e)=\bar{F}(e)+B(e)$，转步骤 3，否则 N 网无可行流。

3. 为了寻求关于 F 的最小费用，我们构造一个赋权有向图 $W(F)$，它的顶点是原 N 网的顶点，而把 N 中的每一条弧 (V_i,V_j) 变成两条相反方向的弧 (V_i,V_j) 和 (V_j,V_i)，定义 $W(F)$ 中的弧的权为：

$$W_{ij} = \begin{cases} -A_{ij} & F_{ij} > B_{ij} \\ 0 & (F_{ij} \leqslant B_{ij}) and (j = i) \\ \infty & (F_{ij} \leqslant B_{ij}) and (j \neq i) \end{cases}$$

（长度为 ∞ 的弧可以从 $W(F)$ 中略去）

然后，在 $W(F)$ 中寻求一条从 V_s 到 V_t 的最短路。

4. 若不存在最短路，则 F 为容量有上下界的最小费用最小流，$\sum A(e) \times F(e)$ 为总输送费用；

若存在最短路，则在原 N 网中得到相应的可改进路 P，这条可改进路的前向弧 $P+$ 为最短路上 F 值大于 B 值的弧的集合，后向弧 $P-$ 为最短路上 F 值小于或等于 B 值的弧的集合。

在可改进路 P 上对 F 进行调整，调整量为：

$$Q = \min[\min_{P+} (F(e) - B(e)), \min_{P-} (C(e) - F(e))]$$

从 t 顶点出发，向 s 顶点倒推，按下述规律缩小可行流：

$$F(e) = \begin{cases} F(e) + Q & e \in P- \\ F(e) - Q & e \in P+ \end{cases}$$

得到新的可行流 F，然后返回步骤 3。

二、求容量有上下界的网络的最小费用最小流程序

```
program bounded_minimum_flow;

    uses
      crt;

    const
      maxn              =  50;

    type
      edgetype          =  record { 网顶点类型 : }
                                 { 单位流量费用、下界、上界、流量、权 }
```

```
                        a,b,c,f,w  :  integer;
                        end;
graphtype         =     xn,1..maxn] of edgetype;{ 网类型 }
  ltype           =     array [1..maxn] of integer;{ 顶点序列类型 }
  settype         =     set of 1..maxn;
 var
  g               :     graphtype;  { 网 }
  l,ll,ft         :     ltype;      { 标号表、辅助表、增广轨 }
  st              :     settype;    { 已检查顶点集合 }
  n,s,t           :     integer;    { 顶点数、源点、汇点 }
  f               :     text;       { 文件变量 }

 procedure init;
  var
    i,j : integer;
    str : string;
  begin
    clrscr;
 write('file name = ');{ 读入文件名并与文件变量连接 }
    readln(str);
     assign(f,str);
    reset(f);{ 读准备 }
    fillchar(g,sizeof(g),0);
    readln(f,n);{ 读入顶点数 }
    while not eof(f) do
      readln(f,i,j,g[i,j].b,g[i,j].c,g[i,j].a);
    { 读入各弧的下界,上界和单位流量的费用 }
    close(f);
   end;

 function min(a,b : integer) : integer;{ 返回 a,b 间的小者 }
  begin
    if a<b then min : =1
           else min : =b
  end;

 procedure ke_xing_liu;
  var
    v,u  : integer;
  begin
    repeat
      fillchar(l,sizeof(l),0);  { 标号表撤空 }
      l[s] : =maxint;           { 从源点出发 }
      st : =[]; ft[s] : =0;{ 已检查的顶点集合置空,从 s 顶点出发搜索增广轨 }
      repeat
        v : =0;
      repeat
        inc(v)
      until (v>n) or (l[v]<>0) and not (v in st);
      if v<=n { v 顶点已标号 }
        then begin
```

```
            for u := 1 to n do
              if l[u]=0 then
          {若与顶点 v 相联的顶点 u 未标号,则置<v,u>前向弧或后向弧标志}
          {并递推可改进量 l[t]}
              if g[v,u].f<g[v,u].c
              then begin
                     l[u] := min(l[v],g[v,u].c-g[v,u].f);
                     ft[u] := v
                   end
              else if g[u,v].f>0
                     then begin
                            l[u] := min(l[v],g[u,v].f);
                            ft[u] := -v
                          end;
             st := st+[v]; { v 进入检查顶点集合 }
            end;
    until (v>n) or (l[t]>0); { 直至无增广轨或寻出增广轨为止 }
    if l[t]>0 { 若增广轨存在,则从汇点 t 倒退,修正轨上各弧的容量 }
      then begin
             u := t;
             repeat
               if ft[u]>0
                 then g[ft[u],u].f := g[ft[u],u].f+l[t]
                 else g[u,-ft[u]].f := g[u,-ft[u]].f-l[t];
               u := abs(ft[u]);
                 until ft[u]=0;
               end
    until v>n
  end;
function ke_xing : boolean;
  var
    i,j : integer;
  begin
    s := n+1; t := n+2; { 增加源点 s̄,汇点 t̄}
    for i := 1 to n do { 构造 N̄ 图 }
      for j := 1 to n do
        if g[i,j].c>0 then
          begin
            g[i,j].c := g[i,j].c-g[i,j].b;
    { 在 N̄ 中保留原边 c̄(e)=c(e)-b(e) }
            g[i,t].c := g[i,t].c+g[i,j].b;
  { 对 G 的每一顶点 i,加新边<i,t̄>,该边上界 c̄(e)=Σb(e) (e∈α(i))}
            g[s,j].c := g[s,j].c+g[i,j].b
  { 对 G 的每一顶点 j,加新边<s̄,j>,该边上界 c̄(e)=Σb(e) (e∈β(j))}
          end;
    g[i,n].c := maxint; g[n,i].c := maxint;
  { 加新边<st>和<ts>,两条边的上界 c̄(e1)=c̄(e2)=∞ }
    inc(n,2);
    ke_xing_liu; { 求 N̄ 的最大流 }
```

```
    dec(n,2);
    for i := 1 to n do
    { 若 s̄ 相连的每条弧的流量≥上界,返回 true;否则返回 false }
      if g[s,i].f<g[s,i].c
        then begin ke_xing := false;exit; end;
    ke_xing := true
  end;
procedure shortest_way_in_w;
{ 求 s 至 t 的最短路径 ft 和 s 至各顶点的最短路径代价 L }
  var
    i,j : integer;
    more: boolean;
  begin
    for i := 1 to n do l[i] := g[s,i].w; { l[i]初始化为 s 至 i 的权 }
    for i := 1 to n do ft[i] := s;
    { 求最短路径 ft 和 s 至各顶点的最短路径代价 l }
    repeat
      ll := l; more := false;
      for i := 1 to n do
        for j := 1 to n do
          if (ll[j]<maxint) and (g[j,i].w<maxint)
            and (ll[j]+g[j,i].w<l[i]) then
              begin
                l[i] := ll[j]+g[j,i].w;
                ft[i] := j; more := true;
              end;
    until not more
  end;
procedure min_cost_flow;
{ 在求出容量有上下界一个可行流的基础上求最小费用最小流 }
  var
    i,j : integer;
  begin
    repeat
      for i := 1 to n do { 构造赋权有向图 w(f) }
        for j := 1 to n do
          if g[i,j].f>g[i,j].b
            then g[i,j].w := -g[i,j].a
            else if i=j
                then g[i,j].w := 0
                else g[i,j].w := maxint;
      shortest_way_in_w; { 求最短路径 ft }
      if l[t]<maxint then { 若最短路存在 }
        begin
          i := t; j := maxint;
  { 从 t 顶点计算调整数 j,j=min[min(f_ij^(k-1)-b_ij),min}(C_ij-f_ij^(k-1))] }
                           u+                   u-
  {
  { u+——最短路径上 f 值>b 值的弧集合 }
```

```
{ u-——最短路径上 f 值≤b 值的弧集合 }
        repeat
          if g[ft[i],i].f>g[ft[i],i].b
            then j:=min(j,g[ft[i],i].f-g[ft[i],i].b)
            else j:=min(j,g[i,ft[i]].c-g[i,ft[i]].f);
          i:=ft[i]
        until i=s;
        i:=t;
        repeat { 从 t 顶点倒推,调整 f(k) }
            if g[ft[i],i].f>g[ft[i],i].b {         ⎧f(k-1)+j   <i,j>∈u- }
              then dec(g[ft[i],i].f,j)     {f(k)=⎨                      }
              else inc(g[i,ft[i]].f,j);    {         ⎩f(k-1)-j   <i,j>∈u+ }
          i:=ft[i];
            until i=s;
          end;
   until l[t]=maxint { 直至无最佳路径为止 }
  end;

procedure main;
  var
    i,j,cost : integer;
  begin
    if not ke_xing { 构造 N'网,求 N'的最大流。若不存在最大流,打印失败信息 }
      then writeln('No ke xing liue')
      else begin
          s:=1; t:=n; { 恢复源点和汇点 }
          g[s,t].c:=0; g[s,t].f:=0; g[t,s].c:=0; g[t,s].f:=0;
          { 撤去<s,t><t,s>两条弧 }
          for i:=1 to n do
            for j:=1 to n do
              begin
                g[i,j].c:=g[i,j].c+g[i,j].b;
                { 恢复原 N 网的上界 }
                if g[i,j].c>0
                  then inc(g[i,j].f,g[i,j].b)
                   { f(e)=f̄(e)+b(e) 作为初始网的一个可行流 }
                  else g[i,j].f:=0
              end;
          min_cost_flow; { 在求出可行流的基础上,求最小费用最小流 }
          cost:=0;
          for i:=1 to n do
          { 打印最小费用最小流的流量,计算流的总输送费用 }
            for j:=1 to n do
              if g[i,j].f>0
                then begin
                       writeln(i,'———',j,' : ',g[i,j].f);
                       inc(cost,g[i,j].f * g[i,j].a)
                     end;
          writeln('total cost : ',cost)
        end;
```

```
    end;
  begin
    init; { 输入含费用和上、下界的网络 N }
    main; { 计算和输出 N 网的最小费用最小流 }
end.
```

6.4.2 一个应用实例——餐厅问题

一、问题及其算法

一个餐厅在相继的 N 天里，第 i 天需要 r_i 块餐巾($i=1,2,\cdots,N$)。餐厅可以购买新的餐巾，每块餐巾费用为 p 分，或者把旧餐巾送到快洗部，洗一块需 m 天，其费用为 f 分，或者送到慢洗部，洗一块需 n 天($n>m$)，其费用为 $s<f$ 分。每天结束时，餐厅必须决定多少块脏的餐巾送到快洗部，多少块送到慢洗部，以及多少保存起来延期送洗。但是洗好的餐巾和购买的新餐巾之和，要满足第 i 天的需求量，并使总的花费最小。

要用回溯法解餐巾问题，其时效非常低。若采用最小费用最小流解题，则简捷得多。首先我们根据题意构造一个新网络 D。

为了安排好 N 天里的餐巾使用计划，我们将一天看作两个顶点，即第 i 天拆成顶点 i 和顶点 i'($1\leqslant i\leqslant N$)，另加一个源点 s 和一个汇点 t，构造出一个具有 $2\times(N+1)$ 个顶点的网络 D。D 网中，每一条弧的上下界 $A(e)$ 和 $B(e)$ 表示某段时间内购买(快洗或慢洗或保存)餐巾的最大值和最小值；流量 $F(e)$ 表示实际餐巾数；单位流量费用 $C(e)$ 表示购买(或送洗)每条餐巾的费用。最初的餐巾使用计划对应了这个网络 D(如图 6-12)：

1. 源点 s 至顶点 i 连一条弧 e，该弧的 $A(e)=0$，$B(e)=F(e)=$ 第 i 天的餐巾用量，$C(e)=$ 每条新餐巾的价值，表示在最初的餐用计划中，前 i 天未洗过或未购买过任何餐巾，第 i 天开始买足新餐巾；

2. 顶点 i 至顶点 i' 连一条弧 e，该弧的 $A(e)=B(e)=F(e)=$ 第 i 天的餐巾用量，$C(e)=0$，表示当天所需的餐巾全部保存起来，延期送洗；

3. 顶点 i 至顶点 t 连一条弧 e，该弧的 $A(e)=0$，$B(e)=\infty$，$C(e)=0$，$F(e)$ 为第 i 天的餐巾用量，表示第 i 天以后不再购买或送洗餐巾；

4. 在顶点 i' 至 $(i+1)'$ 之间连一条弧 e，该弧的 $A(e)=0$，$B(e)=\infty$，$C(e)=0$，表示第 i 天保留一部分餐巾，延至第 $i+1$ 天处理；

5. 若第 i 天快洗的旧餐巾能在最后一天前使用($i+$快洗天数$\leqslant N$)，则在 i' 至顶点($i+$快洗天数)之间连一条弧 e，该弧的 $A(e)=0$，$B(e)=\infty$，$C(e)=$ 快洗每条餐巾的费用，表示第 i 天允许快洗餐巾；

6. 若第 i 天慢洗的旧餐巾能在最后一天前使用($i+$慢洗天数$\leqslant N$)，则在顶点 i' 至($i+$慢洗天数)之间连一条弧 e，该弧的 $A(e)=0$，$B(e)=\infty$，$C(e)=$ 慢洗每条餐巾的费用，表示第 e 天允许慢洗餐巾。

然后从上述的可行流 F 出发，求网络 D 的最小费用最小流。我们可以从弧流量中得出每天餐巾使用的计划。

对于第 i($1\leqslant i\leqslant N$)天来说：

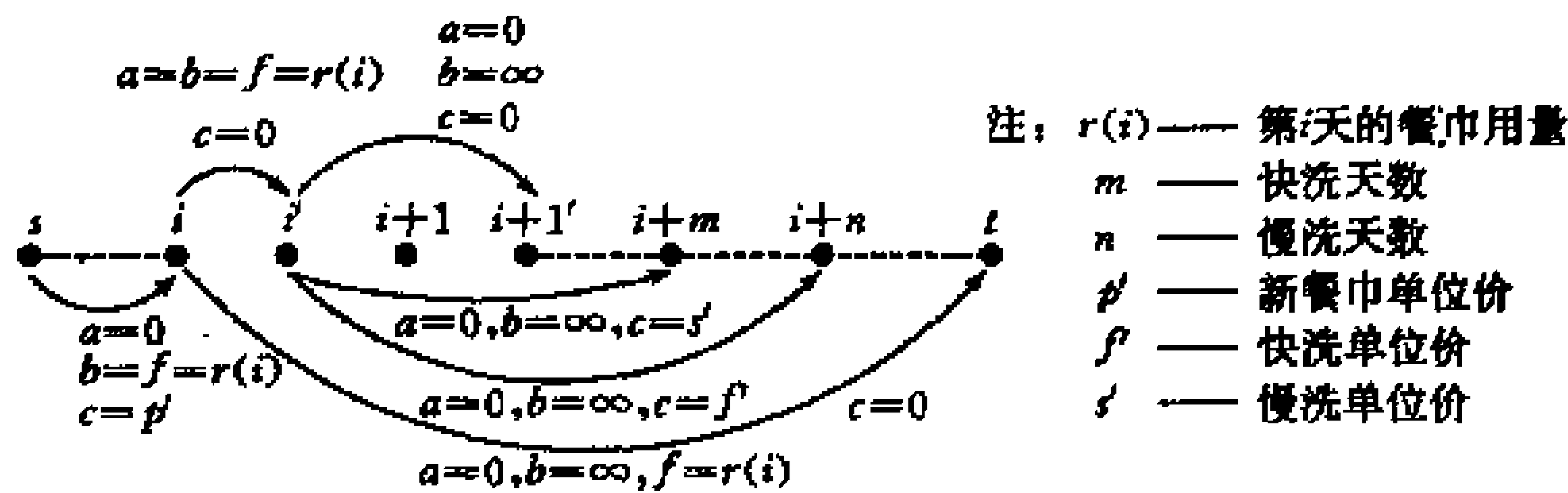


图 6-12

D 网络中顶点 s 至顶点 i 的弧流量表示当天购买的餐巾数；

若 i＋快洗天数≤N，则顶点 i' 至(i＋快洗天数)的弧流量表示当天快洗的餐巾数；

若 i＋慢洗天数≤N，则顶点 i' 至(i＋慢洗天数)的弧流量表示当天慢洗的餐巾数。

顶点 i 至顶点 i' 的弧流量表示当天保存起来延期送洗的餐巾数。

将 N 天中购买的餐巾数×新餐巾的单位价＋快洗的餐巾数×快洗单价＋慢洗的餐巾数×慢洗单价。即为总的花费最小的一种方案。

二、程序题解

```
program napkin;
uses
  crt;
const
  max = 81;
type
  edgetype = record           { 弧类型 }
    a, f, c : byte;           { 下界、流量、上界、单位流量费用，权 }
    b, w : integer;
  end;
  graphtype = array [0..max, 0..max] of edgetype;
  ltype = array [0..max] of integer;
  settype = set of 0..max;
var
  g : graphtype;              { 网 }
  l, ll, h : ltype;           { s 顶点至各顶点的路径代价，辅助变量，最佳路径 }
  r : ltype;                  { 每天的餐巾用量 }
  num, ss, ff, pp, mm, nn : integer;{天数、慢洗费用、快洗费用、新餐巾的价值、快洗天数、漫洗
                                  天数}
  n, s, t : integer;          { 网的顶点个数、源点、汇点 }
  f : text;                   { 文件变量 }
procedure init;
var
  i, j : integer;
  st : string;
begin
```

```
clrscr;
write('Filename > ');
readln(st);                  { 输入文件名 }
assign(f, st);               { 文件名与文件变量连接,读准备 }
reset(f);
readln(f, num);              { 读天数 }
for i := 1 to num do         { 读每天的餐巾用量 }
  read(f, r[i]);
read(f, pp, mm, ff, nn, ss);  { 读新餐巾的价值、快洗餐巾的天数和洗的费用、慢洗餐巾的天
                                数和洗的费用 }
fillchar(g, sizeof(g), 0);   { 邻接矩阵初始化 }
n := num*2+1;   { 0 为源点;2*天数+1 为汇点,构造 N 网 }
s := 0;
t := n;
for i := 1 to num do
begin
  j := i*2-1;
  with g[0, j] do            { 0 至 i*2+1 连一条边 }
  begin
    a := 0;                  { 下界为 0 }
    b := r[i];               { 上界和流量为第 i 天的餐巾用量 }
    c := pp;                 { 单位流量费用为新餐巾的价值 }
    f := r[i];
  end;
  with g[j, j+1] do          { i*2-1 至 i*2 连一条边 }
  begin
    a := r[i];               { 上界、下界和流量为第 i 天的餐巾用量 }
    b := r[i];
    c := 0;                  { 单位流量费用为 0 }
    f := r[i];
  end;
  if i<num then { 若该天非最后一天,则 }
    with g[j+1, j+3] do { i*2 至 i*2+2 连一条边 }
    begin
      a := 0;                { 下界为 0 }
      b := maxint;           { 上界为无穷大 }
      c := 0;                { 单位流量费用为 0 }
    end;
  if i+mm<=num then     { 若该天可以快洗餐巾,则 }
    with g[j+1, 2*(i+mm)-1] do     { i*2 至 i*2+2*快洗天数-1 连一条边 }
    begin
      a := 0;                { 下界为 0 }
      b := maxint;           { 上界为无穷大 }
      c := ff;               { 单位流量费用为快洗的费用 }
    end;
  if i+nn<=num then     { 若该天可以慢洗餐巾,则 }
    with g[j+1, 2*(i+nn)-1] do     { i*2 至 i*2+2*慢洗天数-1 连一条边 }
    begin
      a := 0;                { 下界为 0 }
```

```
        b := maxint;        { 上界为无穷大 }
        c := ss;            { 单位流量费用为慢洗的费用 }
      end;
    with g[j+1, t] do       { i*2至2*天数+1连一条边 }
      begin
        a := 0;             { 上界为0 }
        b := maxint;        { 下界为无穷大 }
        c := 0;             { 单位流量费用为0 }
        f := r[i];          { 流量为第i天的餐巾用量 }
      end;
  end;
  close(f);
end;

function min(a, b : integer) : integer;  { 返回a、b间的小者 }
begin
  if a<b then
    min := a
  else
    min := b;
end;

procedure FindShortestPath;
var
  i, j : integer;
  more : boolean;
begin
  for i := 0 to n do l[i] := g[s, i].w; { L初始化为s至各顶点的权 }
  for i := 0 to n do h[i] := s; { 求最短路径H和s至各顶点的路径代价L }
  repeat
    ll := l;
    more := false;
  for i := 0 to n do
    for j := 0 to n do
      if (ll[j]<maxint) and (g[j, i].w<maxint) and (ll[j]+g[j, i].w<ll[i]) then
      begin
        more := true;
        l[i] := ll[j]+g[j, i].w;
        h[i] := j;
      end;
  until not more;
end;

procedure main;
var
  i, j : integer;
begin
  repeat
    for i := 0 to n do          { 构造赋权图W(F) }
      for j := 0 to n do
        if g[i, j].f>g[i, j].a then
```

```
          g[i, j].w  := -g[i, j].c
        else if g[j, i].f<g[j, i].b then
          g[i, j].w  := g[j, i].c
        else if i = j then
          g[i, j].w  := 0
        else
          g[i, j].w  := maxint;
    findshortestpath;            { 求最短路径 h 和 S 至各顶点的最佳路径代价 }
    if l[t]<0 then               { 最佳路径存在 }
    begin
      i := t;                    { 从 t 顶点倒推调整量 J }
      j := maxint;
      repeat
        if g[h[i], i].f>g[h[i], i].a then
          j := min(j, g[h[i], i].f-g[h[i], i].a)
        else
          j := min(j, g[i, h[i]].b-g[i, h[i]].f);
        i := h[i];
      until i = s;
      i := t;                    { 从 t 顶点倒推,调整 f(k)}
      repeat
        if g[h[i], i].f>g[h[i], i].a then
          g[h[i], i].f  := g[h[i], i].f-j
        else
          g[i, h[i]].f  := g[i, h[i]].f+j;
        i := h[i];
      until i = s;
    end;
  until l[t]>=0;                 { 直至无最佳路径为止 }
end;

procedure show;
var
  i, j, cost : integer;
begin
  cost  := 0;
  for i  := 1 to num do
  begin
    j := i*2;
    writeln('Day ', i, ')');          { 打印第 i 天 }
    writeln('Buy ', g[0, j-1].f, 'new napkins.');
    { 购买新餐巾数--(0,2i-1)的流量 }
    cost  := cost+g[0, j-1].f*pp;
    writeln('Use ', r[i], ' clean napkins.');
    if i+mm<=num then   { 快洗餐巾数——(2i,2(i+mm)-1)的流量 }
    begin
      writeln('Fast wash ', g[j, 2*(i+mm)-1].f, ' napkins.');
      cost  := cost+g[j, 2*(i+mm)-1].f*ff;
    end;
    if i+nn<=num then  { 慢洗餐巾数——(2i,2(i+nn)-1)的流量 }
```

```
    begin
      writeln('Slow wash ', g[j, 2 * (i+nn)-1].f, ' napkins.');
      cost := cost+g[j, 2 * (i+nn)-1].f * ss;
    end;
    if i<num then          { 保存餐巾数——(2i,2(i+1))的流量 }
      writeln('Leave ', g[j, j+2].f, ' dirty napkins.');
    writeln('Press <ENTER> to continue...');
    readln;
  end;
  writeln('Total cost : ', cost);
end;

begin
  init; { 输入数据,构造N网 }
  main; { 通过求N网的最小费用最小流,得出最佳方案 }
  show; { 输出结果 }
end.
```

6.5 求有供需约束的可行流

一、问题及其分析

图 6-13 是一种商品供应系统,弧的方向表示商品流向,弧上的数字表示商品从某地至某地的最大通过能力。商品的货源来自于 X_1,X_2,X_3,其中 X_1 的供应量为 5,X_2 为 10,X_3 为 5。这些商品在商场 Y_1,Y_2,Y_3 出售。Y_1 的需求为 5,Y_2 为 10,Y_3 为 5。问是否所有的需求皆可同时被满足?

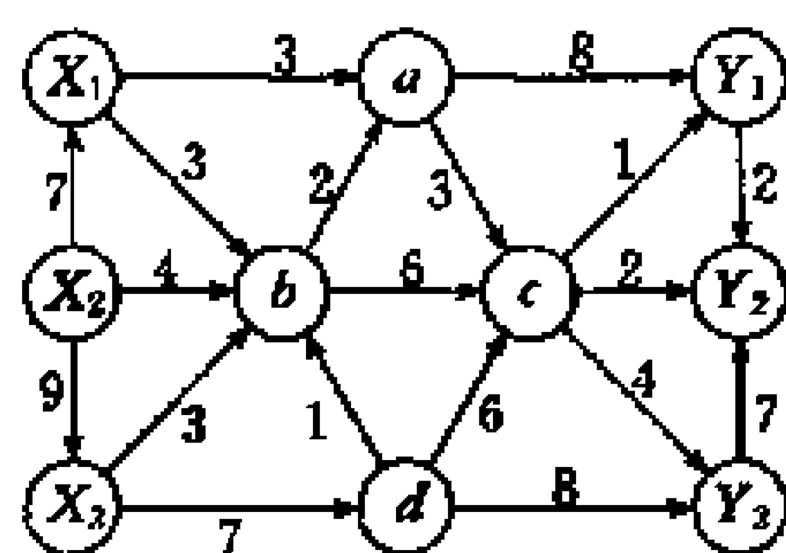


图 6-13

上述网络是一种特殊的网络。在这个网络中,源点和汇点都不止一个。对于每一个源点 $X_i(1\leqslant i\leqslant 3)$来说,存在一个供应量;对于每一个汇点 $Y_j(1\leqslant j\leqslant 3)$来说,也存在一个需求量。题目所要求的所有汇点的需求量是否同时被满足的问题,可以归结为以下数学模型:

N 是一个网络,每条弧 e 对应一个弧容量 $C(e)$,又顶点集合 $X=\{X_1,X_2,\cdots,X_m\}$是源集合,$A(X_i)$是源点 X_i 的供应量;顶点集合 $Y=\{Y_1,Y_2,\cdots,Y_n\}$是汇集合,$P(Y_i)$是汇点的需求量。现在,要求出满足下列条件

(1) 对于所有弧 e,$0\leqslant F(e)\leqslant C(e)$;

(2) 对于除去源点和汇点的中间顶点 U, $\sum\limits_{e\in\beta(U)}F(e)-\sum\limits_{e\in\alpha(U)}F(e)=0$

(3) $\sum\limits_{e\in\alpha(X_i)}F(e)-\sum\limits_{e\in\beta(X_i)}F(e)\leqslant A(X_i)$ $\quad(1\leqslant i\leqslant N)$

即对于产地,求不能过于供:

(4) $\sum\limits_{e\in\beta(Y_i)}F(e)-\sum\limits_{e\in\alpha(Y_i)}F(e)\geqslant P(X_i)$ $\quad(1\leqslant i\leqslant N)$

即对于销地,供不能低于求的流 F,这个流称为满足供需约束的可行流。

下面，给出上述问题的算法：

1. 构作新网络 N'

(1) 新增两个顶点 X_0, Y_0；

(2) 加 m 条以 X_0 为尾的弧 $(X_0,X_1),(X_0,X_2),\cdots,(X_0,X_m)$，每条弧的容量为 $C(X_0,X_i)=A(X_i), i=1,2,\cdots,m$；

(3) 加 n 条以 Y_0 为头的弧 $(Y_1,Y_0),(Y_2,Y_0),\cdots,(Y_n,Y_0)$，每条弧的容量为 $C(Y_i,Y_0)=P(Y_i), i=1,2,\cdots,n$；

(4) 以 X_0 为 N' 的源点，Y_0 为 N' 的汇点。

2. 求 N' 的最大流 F

若 F 使得流入 Y_0 的每条弧都满载(即 $C(Y_i,Y_0)=F(Y_i,Y_0)$，其中 $i=1,2,\cdots,N$)，则 F 是原网络 N 中满足供需约束的一个可行流。

二、求可行流程序

```
program flow;
    { P191 }
    { G15.dat t.dat}
    const
      maxn          =  30;
    type
      nodetype      =  record { 增广轨顶点类型： 标号标志，检查标志 }
                         l,p : integer;
                       end;
      arctype       =  record { 网顶点类型： 容量，流量 }
                         c,f : integer
                       end;
      gtype         =  array [1..maxn,1..maxn] of arctype;  { 网类型 }
      ltype         =  array [1..maxn] of nodetype;       { 增广轨类型 }
      settype       =  set of 1..maxn;
    var
      lt            :  ltype;                 { 增广矩阵 }
      g             :  gtype;                    { 网 }
      v,s,t,m,n,       { 顶点数、源点、汇点、货源数、市场数 }
      dem           :  integer;         { 货源的供应总量 }
      f             :  text;                      { 文件变量 }
      x,y           :  settype;          { 货源集合，市场集合 }
      a,b           :  array [1..maxn] of integer;
                         { 各货源的供应量，各市场的需求量 }
    procedure read_graph;
      var
        str : string;
        i,j : integer;
      begin
        write('Graph file = '); { 读入文件名并与文件变量连接 }
        readln(str);
```

```
    assign(f,str);
    reset(f);                    { 读准备 }
    readln(f,v);                 { 读入顶点数 }
    fillchar(g,sizeof(g),0);  { 网初始化 }
    fillchar(lt,sizeof(lt),0);  { 增广矩阵初始化 }
    for i:=2 to v+1 do { 读入初始网 }
      for j:=2 to v+1 do read(f,g[i,j].c);
    readln(f,m,n); { 读入货源和市场数 }
    x:=[];
    { 读入每个货源的供应量。注:第i个货源的供应量存入 a[i+1] }
    { x 存放 a 的下标集合. 因为在新网络中,所有顶点的序号增 1,以新增一个源 X0}
    for i:=1 to m do
      begin
        read(f,j,a[j+1]);x:=x+[j+1]
      end;
    y:=[];dem:=0;
    { 读入每个市场的需求量: 注第i个市场的需求量存入 b[i+1] }
    { y 存放 b 的下标集合,因为在新图中,所有顶点序号增 1 }
    for i:=1 to n do
      begin
        read(f,j,b[j+1]);y:=y+[j+1];
        dem:=dem+b[j+1]
      end;
    close(f);
  end;

function check:integer; { 返回一个已标号而未检查的顶点序号 i }
  var
    i:integer;
  begin
    i:=s;
    while (i<=v) and not ((lt[i].l<>0) and (lt[i].p=0)) do inc(i);
    if i>v then check:=0
          else check:=i
  end;

function ford(var a:integer):boolean; { 寻找增广轨和改进量 a }
  var
    i,j,m,x:integer;
  begin
    ford:=true;
    fillchar(lt,sizeof(lt),0);
    lt[s].l:=1;
    repeat
      i:=check;
      if i=0 then exit;
      for j:=s to v do
        if (lt[j].l=0) and ((g[i,j].c<>0) or (g[j,i].c<>0))
          then begin
                  if (g[i,j].f<g[i,j].c) then lt[j].l:=i;
                  if (g[j,i].f>0) then lt[j].l:=-i
```

```
              end;
          lt[i].p:=1
        until (lt[t].l<>0);
        m:=t;a:=maxint;
        repeat
          j:=m;m:=abs(lt[j].l);
          if lt[j].l<0 then x:=g[j,m].f;
          if lt[j].l>0 then x:=g[m,j].c-g[m,j].f;
          if a>x then a:=x
        until m=s;
        ford:=false
      end;
  procedure fulkerson(a:integer); { 修正增广轨上的流量 }
    var
      m,j:integer;
    begin
      m:=t;
      repeat
        j:=m;m:=abs(lt[j].l);
        if lt[j].l<0 then g[j,m].f:=g[j,m].f-a;
        if lt[j].l>0 then g[m,j].f:=g[m,j].f+a
      until m=s
    end;
  procedure answer;
    { 若s顶点至各货源的可行流满载,则表明所有需求同时满足,}
    { 打印每条弧的可行流; 否则打印无解 }
    var
      i,j:Integer;
    begin
      j:=0;
      for i:=1 to v do j:=j+g[1,i].f;
      if j<>dem
        then writeln('No solution ')
        else for i:=2 to v-1 do
               for j:=2 to v-1 do
                 if g[i,j].f<>0 then writeln(i-1,'————',j-1,' ',g[i,j].f)
end;
  procedure proceed;
    var
      i,del:integer;
      success:boolean;
    begin
      s:=1;v:=v+2;t:=v; { 新增源点s=1,汇点t=v+2 }
      for i:=2 to v-1 do { 新增s到所有货源的边,弧的容量为a(i) }
        if i in x then g[s,i].c:=a[i];
      for i:=2 to v-1 do
        if i in y then g[i,t].c:=b[i];
        { 新增所有市场到t的边,弧的容量为b(i) }
      repeat
```

```
      success := ford(del); { 在新网中寻找增广轨和可改进量 del }
      if success then answer { 若无增广轨,则输出结果 }
              else fulkerson(del) { 否则修正增广轨的流量 }
    until success; { 直至新网的最大流求出为止 }
  end;

begin
  read_graph;  { 输入含供需约束的网 }
  proceed      { 输出满足需求的方案 }
end.
```

6.6 求图的连通度

一、问题及其分析

我们曾在第四章"4.1 连通性的基本概念和定义"一节中给出图的连通度的定义：就是一个具有 N 个顶点的图 G，在去掉任意 $k-1$ 个顶点后($1\leqslant k\leqslant N$)，所得的子图仍连通，而去掉 k 个顶点后的子图不连通，则称 G 是 k 连通图，k 称作图 G 的连通度，记作 $k(G)$。

这一节，我们将给出求图的连通度的算法。首先，讲述一下独立轨的概念：

A,B 是无向图 G 的两个顶点，我们称为从 A 到 B 的两两无公共内顶的轨为独立轨(或者说，A,B 是有向图 G 的两个顶点，称从 A 到 B 的两两无公共内顶的有向轨为有向独立轨)，A 到 B 独立轨的最大条数(A 到 B 有向轨的最大条数)，记作 $P(A,B)$。

(注：( )内为有向图的补充定义。)

例如，图 6-14 的一个具有 7 个顶点的连通图，从顶点 1 到顶点 3 有 3 条独立轨，即 $P(1,3)=3$

1—2—3

1—7—3

1—6—5—4—3

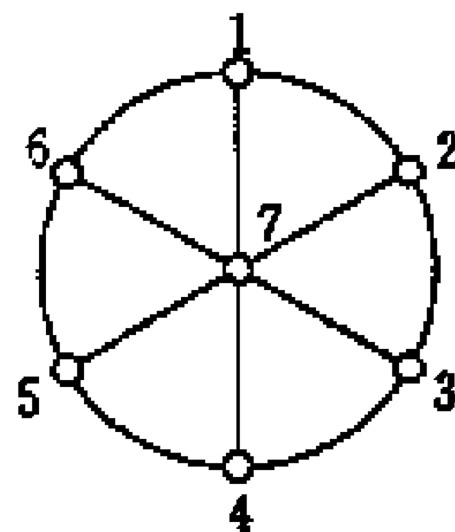


图 6-14

如果我们分别从这 3 条独立轨中，每条轨抽出一个内点，在 G 图中删掉，则图不连通。例如去掉顶点 2,7,6，或者去掉顶点 2,7,5，或者去掉顶点 2,7,4。

读者很容易得出 $P(2,4)=P(2,5)=P(2,6)=P(3,5)=P(3,6)=P(4,6)=P(4,1)=3$。即每两个不相邻的顶点间，都最多有 3 条独立轨。每轨任删一个内点，也会使图 G 变成不连通。显然 $k(G)=3$。若连通图 G 的两两不相邻顶点间的最大独立轨数不尽相同，则最小的 $P(A,B)$ 值，即为 $k(G)$。

$$k(G)=\begin{cases}\text{顶点数}-1 & G\text{ 为(双向)完全图}\\ \min\limits_{A,B\text{不相邻}} P(A,B) & G\text{ 非(双向)完全图}\end{cases}$$

(注：双向完全图，指有向图的任两个顶点互相可达。)

那么，如何求不相邻的两个顶点 A,B 间的最大独立轨数 $P(A,B)$，应删去图中哪些顶点(被删顶点数$=P(A,B)$)后使图不连通呢？我们采用求最大流的办法来解决这个问题：

1. 构造一个网络 N

若 G 为无向图：

(1) 原 G 图中的每个顶点 V 变成 N 网中的两个顶点 V' 和 V''，顶点 V' 至 V'' 有一条弧连接，弧容量为 1；

(2) 原 G 图中的每条边 $e=UV$，在 N 网中有两条弧 $e'=U''V'$，$e''=V''U'$ 与之对应，e' 弧容量为∞，e'' 弧容量为∞；

(3) A'' 为源顶点，B' 为汇顶点

若 G 为有向图(见图 6-15)：

(1) 原 G 图中的每个顶点 V 变成 N 网中的两个顶点 V' 和 V''，顶点 V' 至 V'' 有一条弧连接，弧容量为 1；

(2) 原 G 图中的每条弧 $e=UV$ 变成一条有向轨 $U'U''V'V''$，其中轨上的弧 $U''V'$ 的容量为∞；

(3) A'' 为源顶点，B' 为汇顶点

2. 求 A'' 到 B' 的最大流 F

3. 流出 A'' 的一切弧的流量和 $\sum_{e\in(A'',V)} F(e)$，即 $P(A,B)$。所有具有流量 1 的弧 (V',V'') 对应的 V 顶点组成一个割顶集，在 G 图中去掉这些顶点则 G 图变成不连通。

有了求 $P(A,B)$ 的算法基础，我们不难得出 $k(G)$ 的求解思路：

首先设 $k(G)$ 的初始值为∞，然后分析图的每一对顶点。如果顶点 A,B 不相邻，则用求最大流的方法得出 $P(A,B)$ 和对应的割顶集。若 $P(A,B)$ 为目前最小，则记当前的 $k(G)$ 值为 $P(A,B)$，保存其割顶集。直至所有不相邻的顶点对分析完为止，就可得出图的顶连通度 $k(G)$ 和最小割顶集了(见图 6-16)。

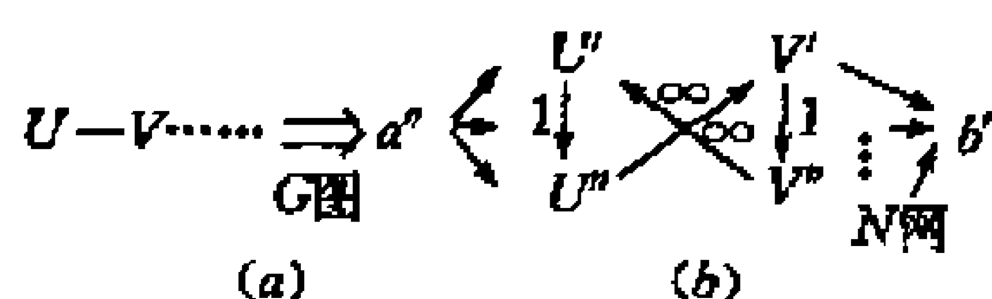


图 6-15

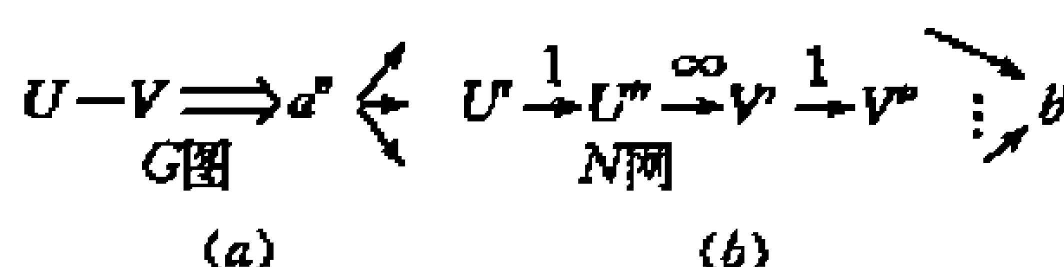


图 6-16

我们可以从图的连通度中，引出一些有趣的结论：

1. G 是 k 的连通图，$k\geqslant 2$，则任意 k 个顶点共圈；

2. G 是 k 的连通图的充要条件是 $k+1\leqslant G$ 的顶点数 N，从图中任取一个含 k 个顶点的子集 U 和 U 集外的一个顶点 X，则存在 $|U|$ 条具有下列性质的轨：

(1) 每条轨的一个端顶点为 X，另一个端顶点在 U 中；

(2) 两两轨间，除 X 外再无公共顶点。

二、求图的连通度的程序

```
program din_lian_tong_du;
{ 求图的连通度 }
```

```
uses
  crt;
const
  maxn              =  100;
type
  edgetype          =  record { 网顶点类型： 容量,流量 }
                         c,f : integer;
                       end;
  graphtype         =  array[1..maxn,1..maxn] of edgetype; { 网类型 }
  ltype             =  array[1..maxn] of integer; { 顶点序列类型 }
  settype           =  set of 1..maxn;
var
  g                 :  graphtype;  { 网 }
  l,ft              :  ltype;      { 标号,弧的前向、后向标志 }
  st,way            :  settype;    { 已检查的顶点集合, 割顶集 }
  n,s,t             :  integer;    { 顶点数,源点,汇点 }
  f                 :  text;       { 文件变量 }
procedure init;
  var i,j,k : integer;
      str : string;
    begin
      clrscr; write('filename='); readln(str);
      { 读入文件名,并与文件变量连接 }
      assign(f,str); reset(f);  { 读准备 }
      readln(f,n);              { 读入顶点数 }
      readln(f,k);
      {    ┌1      有向图 }
      {k=─┤               }
      {    └2      无向图 }
      fillchar(g,sizeof(g),0); { 网初始化 }
      for i := 1 to n do { 读入图,构作新图 g=(V,E) }
        g[i,i+n].c := 1;
      while not eof(f) do
        begin
          readln(f,i,j);
          g[i+n,j].c := maxint;
          if k=2
            then g[j+n,i].c := maxint;
        end;
      close(f);
    end;
function min(a,b : integer) : integer; { 返回 a,b 间的小者 }
  begin
    if a<b then min := a
           else min := b;
  end;
procedure ford_fulkerson;
  var v,u : integer;
```

```
begin
  repeat fillchar(l,sizeof(l),0);
         l[s] : =maxint;
         st : =[]; ft[s] : =0;
         repeat v : =0; { 寻找一个已标号而未检查的顶点序号 v }
                repeat inc(v);
                until (v>n) or (l[v]>0) and not (v in st);
                if v<=n then
  { 寻找所有与顶点 v 连接且未标号的顶点 u,弧<v,u>置前向标志或后向标志 }
                   begin
                     for u : =1 to n do
                       if l[u]=0 then
                         if g[v,u].f<g[v,u].c
                           then begin
                                  l[u] : =min(l[v],g[v,u].c-g[v,u].f);
                                  ft[u] : =v;
                                end
                           else if g[u,v].f>0
                                then begin
                                       l[u] : =g[u,v].f;
                                       ft[u] : =-v;
                                     end;
                     st : =st+[v]; { v 进入已检查顶点集合 }
                   end;
         until (v>n) or (l[t]<>0);
    { 直至无增广轨或找出了 s 至 t 的增广轨为止. 该轨的可改进量为 l[t] }
         if l[t]<>0 then
    { 若增广轨存在,则从顶点 t 开始,倒向调整轨上的流量 }
         begin
           u : =t;
           repeat if ft[u]>0
                  then g[ft[u],u].f : =g[ft[u],u].f+l[t]
                  else g[u,abs(ft[u])].f : =g[u,abs(ft[u])].f-l[t];
                  u : =abs(ft[u]);
           until ft[u]=0;
         end;
  until v>n; { 直至无增广轨为止 }
end;
procedure main;
  var a,b,i,j,k,x,m : integer;
    begin
      k : =maxint; m : =n; n : =n * 2;
      for a : =1 to m do
        for b : =1 to m do
          if (a<>b) and (g[a+m,b].c=0) then
            { 若<a,b>不在图中 }
              begin s : =a+m; t : =b; { 以 a 为源,b 为汇,求最大流 P(a,b) }
                    for i : =1 to n do
                      for j : =1 to n do
```

```
                g[i,j].f := 0;
              ford_fulkerson;
              x := 0;
              for i := 1 to n do { 累计 P(a,b) }
                inc(x,g[s,i].f);
              if x<k { 若 P(a,b)为目前最小,则记下 }
                then begin
                        k := x;
                        way := [];
                        for i := 1 to m do
                { 若<vi,vi+n>的流量为 1,则对应的 vi 为割顶,进入割顶集 }
                          if g[i,i+m].f=1
                          then way := way+[i];
                        end;
          end;
      if k=maxint then { 若为完全图,割顶集为[1..m-1] }
        begin
          k := m-1;
          way := [1..m-1];
        end;
      writeln('ding lian tong du = ',k); { 打印连通度 k 和割顶集 }
      for i := 1 to m do
        if i in way then writeln('take din ',i);
    end;

begin
  init; { 输入图,构作新网 }
  main; { 计算和输出连通度、割顶集 }
end.
```

6.7 求图的边连通度

一、问题及其分析

我们在 4.1 节中关于连通性的基本概念和定义中给出图的边连通度的定义:即具有 $|e|$ 条边的图 G,任意去掉 $k-1(1\leqslant k\leqslant|e|)$ 条边后,所得的子图仍连通,而去掉 k 条边后的子图不连通,则称 G 是 k 边连通图。k 称作图 G 的边连通度,记作 $k'(G)$。

这一节中,我们将给出求图边连通度的算法。首先,讲述一下弱独立轨的概念:

A,B 是无向图 G 的两个顶点,我们称从 A 到 B 的两两无公共边的轨为弱独立轨(或者说,A,B 是有向图 G 的两个顶点,称从 A 到 B 的两两无公共边的有向轨为有向弱独立轨)。A 到 B 的弱独立轨的最大条数(A 到 B 的有向弱独立轨的最大条数)记作$P'(A,B)$。

(注:()内为有向图的补充定义。)

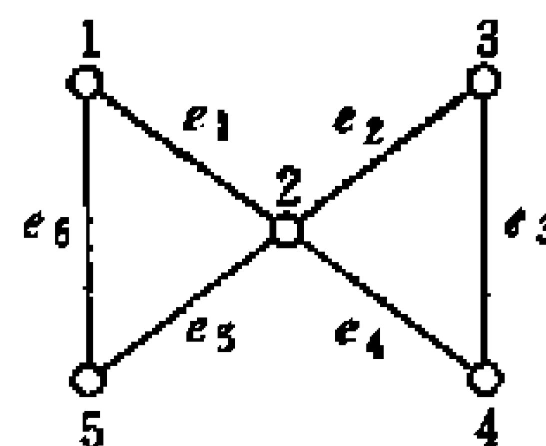


图 6-17

例如,图 6-17 是一个具有 6 条边的连通图,从顶点 1 到顶点 4 有 2 条弱独立轨,即 $P'(1,4)=2$:

$$(e_1,e_2,e_3)$$

$$(e_4,e_5,e_6)$$

如果我们分别从这两条弱独立轨中，每轨取一条边，在 G 图中去掉，则图不连通。例如去掉 e_1,e_6，或去掉 e_1,e_5，或去掉 e_3,e_4 等。

读者很容易得出 $P'(1,3)=P'(5,3)=P'(5,4)=2$，即每两个不相邻的顶点间，都最多有 2 条弱独立轨，每轨任删一条边，亦会使图 G 变成不连通，显然 $k'(G)=2$。若连通图 G 的两两不相邻顶点间的最大弱独立轨数不尽相同，则最小的 $P'(A,B)$ 值即 $k'(G)$。

$$k'(G)=\begin{cases}\text{顶点数}-1 & G\text{ 为(双向)完全图}\\ \min\limits_{A,B\text{不相邻}} P'(A,B) & G\text{ 非(双向)完全图}\end{cases}$$

那么，不相邻的两个顶点 A,B 间最大弱独立轨数 $P'(A,B)$ 如何求得，应删去图中哪些边(被删去边数 $=P'(A,B)$)后使图不连通呢？我们还是采用最大流的方法来解决这个问题：

1. 构造一个网络 N

若 G 为无向图：

(1) 原 G 图中的每条边 $e=UV$ 变成重边，再定向为互为反向，得 $e'=UV,e''=VU$，e' 的容量为 1，e'' 的容量亦为 1；

(2) 以 A 为源顶点，B 为汇顶点

若 G 为有向图：

(1) 原 G 图中每条弧的容量为 1；

(2) 以 A 为源顶点，B 为汇顶点。

2. 求 A 和 B 的最大流 F

3. 流出 A 的一切弧的流量和 $\sum\limits_{e\in(A,V)} F(e)$，即 $P'(A,B)$，流出 A 的流量为 1 的弧 (A,V) 组成一个桥集，在 G 图中删去这些弧，则 G 图变成不连通。

有了求 $P'(A,B)$ 的算法基础，我们不难得出 $k'(G)$ 的求解思路：

首先设 $k'(G)$ 的初始值为 ∞，然后分析图的每一对顶点。如果顶点 A,B 不相邻，则用求最大流的方法得出 $P'(A,B)$ 和对应的桥集。若 $P'(A,B)$ 为目前最小，则记当前的 $k'(G)$ 为 $P'(A,B)$，保存其边桥集，直至所有不相邻的顶点对分析完为止，即可得出图的边连通度 $k'(G)$ 和最小桥集了。

我们可以从图的边连通度中引出一些有趣的结论：

1. A 是有向图 G 的一个顶点，若顶点 A 与 G 的其它所有顶点 V 间的 $P'(A,V)$ 的最小值为 k，即 $\min\limits_{V\in U-\{A\}} P'(A,V)=k$，则 G 中存在以 A 为根的 k 棵无公共边的外向生成树；

2. 设 G 是有向图，$0<k\leqslant k'(G)$，L 是 0 至 k 之间任意一个整数。对于图 G 的任一对顶点 UV 来说，存在 U 到 V 的 L 条弱独立有向轨，同时存在 V 到 U 的 $k-L$ 条弱独立有向轨

二、求图边连通度的程序

```
program bian_lian_tong_du;
```

```
uses
  crt;

const
  maxn              =  50; { 最多顶点数 }

type
  edgetype          =  record { 网顶点类型：容量、流量 }
                         c,f : integer;
                       end;
  graphtype         =  array[1..maxn,1..maxn] of edgetype; { 网类型 }
  ltype             =  array[1..maxn] of integer;        { 增广轨类型 }
  settype           =  set of 1..maxn;

var
  g                 :  graphtype;  { 网 }
  l,ft              :  ltype;      { 标号表，前向弧后向弧标记 }
  st,way            :  settype;    { 已检查顶点集合，桥集 }
  n,s,t,d,best_s    :  integer;
  { 顶点数、源点，顶点，有向、无向标志、辅助变量 }
  f                 :  text;      { 文件变量 }

procedure init;
  var i,j : integer;
      str : string;
    begin
      clrscr; write('filename='); readln(str);
      { 读入文件名，并与文件变量连接 }
      assign(f,str); reset(f);     { 读准备 }
      readln(f,n);                 { 读入顶点数 }
      fillchar(g,sizeof(g),0);     { 网初始化 }
      readln(f,d); {     ┌1           有向图 }
                   {d=<                     }
                   {     └2           无向图 }
      while not eof(f) do { 读入各弧的参数 }
        begin
          readln(f,i,j);
          g[i,j].c : =1;
          if d=2 then g[j,i].c : =1; { 若为无向图，则矩阵对称 }
        end;
      close(f);
    end;

function min(a,b : integer) : integer; { 返回 a、b 间的小者 }
  begin
    if a<b then min : =a
           else min : =b;
  end;

procedure ford_fulkerson;
  var v,u : integer;
    begin
      repeat fillchar(l,sizeof(l),0); { 增广轨初始化 }
```

```
        l[s]:=maxint;           { 从 s 出发搜索增广轨 }
        st:=[]; ft[s]:=0;
        repeat v:=0;          { 搜索已标号但未检查的顶点 v }
              repeat inc(v);
              until (v>n) or (l[v]>0) and not (v in st);
              if v<=n then
        { 若连接 v 的另一端点 u 未标号,则置<v,u>正向弧或反向弧标记 }
                    begin
                      for u:=1 to n do
                        if l[u]=0 then
                          if g[v,u].f<g[v,u].c
                            then begin
                                   l[u]:=min(l[v],g[v,u].c-g[v,u].f);
                                   ft[u]:=v;
                                 end
                            else if g[u,v].f>0
                                   then begin
                                          l[u]:=g[u,v].f;
                                          ft[u]:=-v;
                                        end;
                      st:=st+[v]; { v 进入检查顶点集合 }
                    end;
        until (v>n) or (l[t]<>0);
        { 直至无增广轨或改进量为 l[t]的增广轨存在为止 }
        if l[t]<>0 then { 若增广轨存在,则修改该轨上的流量 }
          begin
            u:=t;
            repeat if ft[u]>0
                     then g[ft[u],u].f:=g[ft[u],u].f+l[t]
                     else g[u,abs(ft[u])].f:=g[u,abs(ft[u])].f-l[t];
                     u:=abs(ft[u]);
            until ft[u]=0;
          end;
    until v>n; { 直至无增广轨为止 }
  end;

procedure main;
  var i,j,k,x : integer;
    begin
      k:=maxint;
      for s:=1 to n do
        for t:=1 to n do
          if s<>t then
            begin
              for i:=1 to n do  { 流量初始化 }
                for j:=1 to n do
                  g[i,j].f:=0;
              ford_fulkerson; { 求 P'(s,t) }
              x:=0;            { 求边连通度 x }
              for i:=1 to n do
```

```
            inc(x,g[s,i].f);
          if x<k then
          { x 为目前最小,则记下目前最佳边连通度、源点 s 和 }
          { 与 s 相连的所有流量为 1 的弧端点 }
            begin
              k := x;
              best_s := s;
              way := [];
              for i := 1 to n do
                if g[s,i].f=1 then way := way+[i];
            end;
        end;
    writeln('bian lian tong du : ',k); { 打印边连通度 k 和桥集 }
    for i := 1 to n do
      if i in way then writeln('take ',best_s,'--',i);
  end;

begin
  init; { 输入图 }
  main; { 计算并输出边连通度和桥集 }
end.
```

第七章　匹配问题

7.1　匹配的基本概念

在 1.1 节引言中，我们曾经讲过一个飞行驾驶员的匹配问题，那里也介绍了“匹配”这一概念，这一章就专门研究与匹配问题有关的算法。首先让我们把匹配的定义再明确地讲一下。

设 $G=[V,E]$ 是一个无向图，$M\subseteq E$ 是 G 的若干条边的集合，如果 M 中的任意两条边都没有公共的端点，就称 M 是一个匹配。

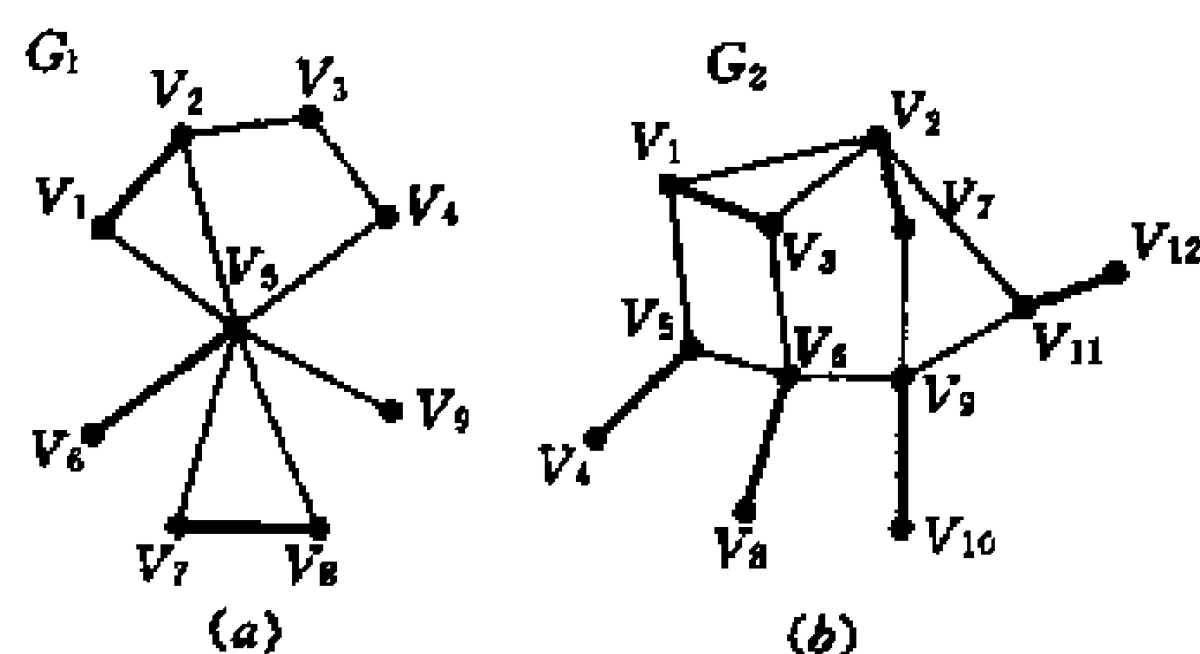


图　7-1

例如图 7-1(a)、(b)中的粗线组成的边集合 M_1 与 M_2 分别是图 G_1 与 G_2 的匹配。

读者稍加分析后便可得出飞行员匹配问题的数学模型：

从给定的图 $G=[V,E]$ 的所有匹配中，把包含边数最多的匹配找出来。这种匹配即所谓的最大匹配问题。

根据图的类型不同，匹配问题也可分为两种：

1. 二分图的匹配

二分图是这样一种图：G 的顶点集合 V 分成两部分 X 与 Y，G 中每条边的两个端点一定是一个属于 X 而另一个属于 Y。例如飞行员分成两部分，一部分是正驾驶员，一部分是副驾驶员。显然，如何搭配正副驾驶员才能使出航飞机最多的问题可以归结为一个二分图上的最大匹配问题。求二分图最大匹配的算法是以 6.1 节中讲的求网络最大流的方法为基础的。

2. 任意图的最大匹配

求任意图的最大匹配(或者二分图的其它形式的匹配，如完备匹配、最佳匹配)，是否也可归结成网络问题来解决？有不少人试验过，但没有成功，因此要另找办法。经过一些数学家的研究，发现要解决上述匹配问题，必须先找出一些研究匹配理论的工具。下面我们将要介绍的支错轨、可增广轨、交错树、带花树，就是这样一些工具。

下面几个定义，都是相对于图 G 的某一个给定的匹配 M 来说的。

未盖点

设 V_i 是图 G 的一个顶点，如果 V_i 不与任意一条属于匹配 M 的边相关联，就称 V_i 是一个未盖点。

拿图 7-2(a)中的图与由粗线组成的匹配 M 来说，$V_2,V_1,V_8\cdots$ 都是未盖点。

为什么取名未盖点呢？直观地说，在给定了一个匹配 M 以后，我们可以认为每一条

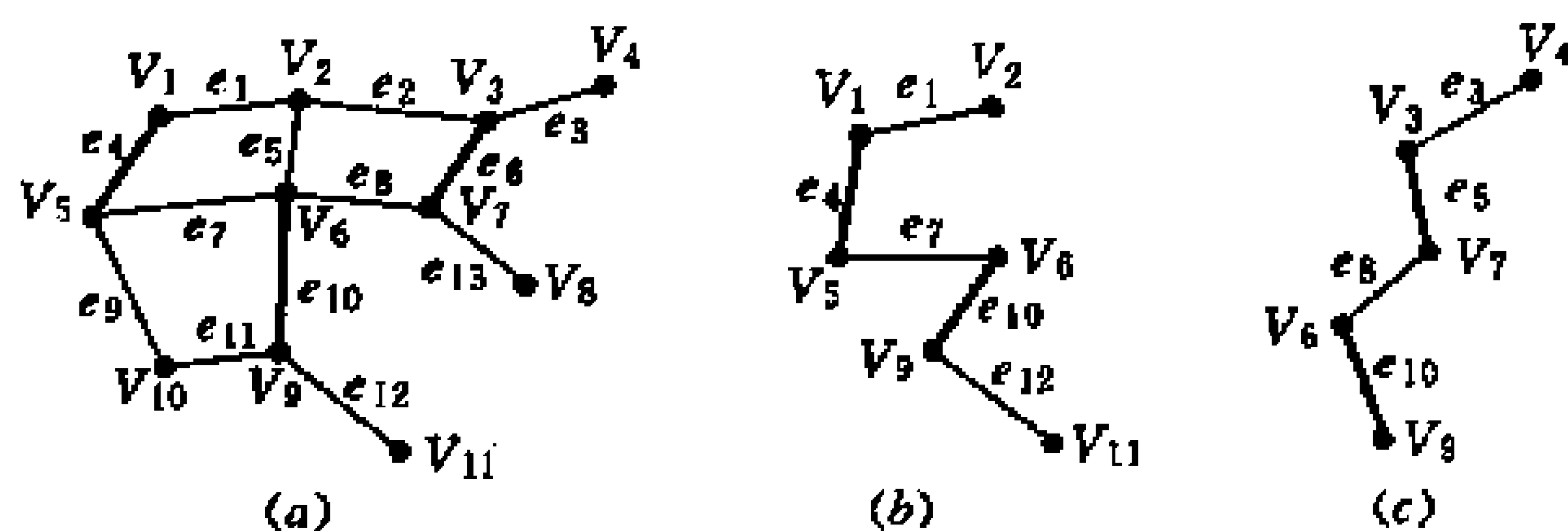


图 7-2

属于 M 的边 E_i“盖住”了两个顶点，就是 E_i 的两个端点。例图 7-2(a)中的 E_4 盖住了 V_1 与 V_5，E_8 盖住了 V_3 与 V_7，等等。而其余顶点，也就是没有被盖住的顶点，自然就称为未盖点了。

交错轨

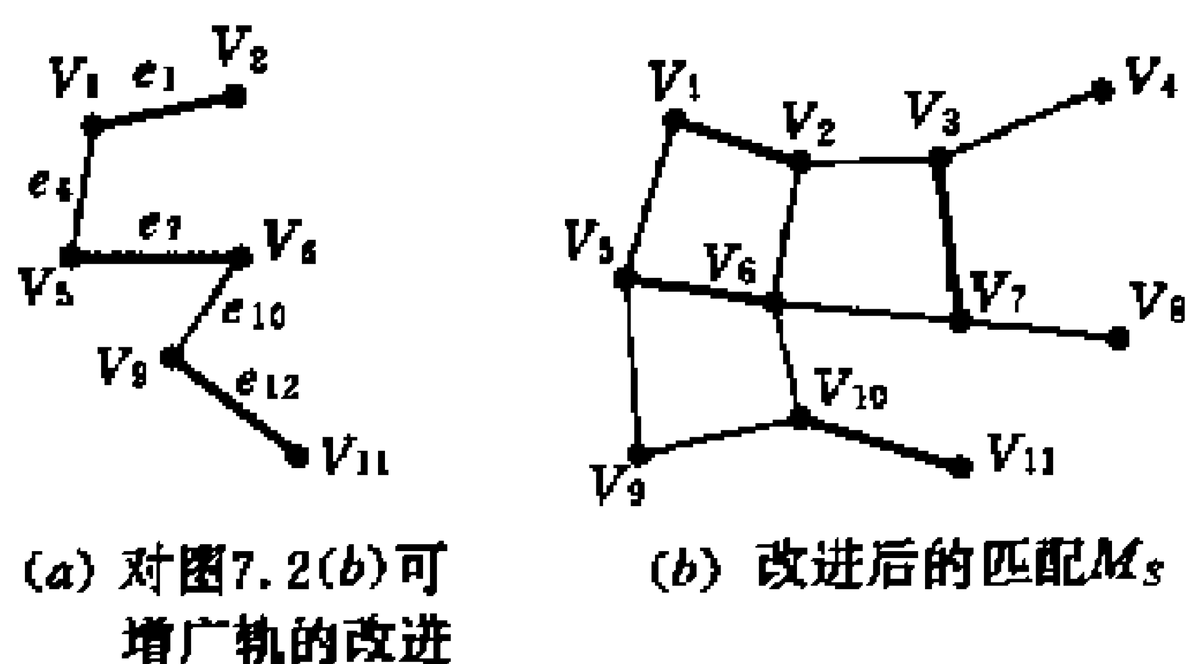
设 P 是图 G 的一条轨，如果 P 的任意两条相邻的边一定是一条属于 M 而另一条不属于 M，就称 P 是一条交错轨。图 7-2(b)，(c)是从图 7-2(a)中找出来的交错轨。顺便说一下，如果轨 P 仅含一条边，那么无论该边是否属于匹配 M，P 一定是一条交错轨。

可增广轨

两个端点都是未盖点的交错轨叫做可增广轨。

拿图 7-2(b)，(c)中的两条交错轨来说，(b)中的那一条轨的两个端点是未盖点 V_2 与 V_{11}，因此这条轨是可增广轨，而(c)中的那条交错轨不是可增广轨。另外，如果图中发现了两个未盖点之间仅含一条边，那么单单这一条边就组成了一条可增广轨了，为什么？因为前面已经说过，单单这条边就组成一条交错轨，再加上它的端点都是未盖点，当然就是一条可增广轨了。

为什么要用“可增广轨”这样一个名字呢？原来对于 G 的一个匹配 M 来说，如果能够找到一条可增广轨，那么这个匹配 M 一定可以通过下述方法改进成一个多包含一条边的匹配 M_S(即 M 匹配扩充了)：



(a) 对图7.2(b)可增广轨的改进

(b) 改进后的匹配M_S

图 7-3

把 P 中原来属于匹配的边从匹配 M 中去掉(粗边改成细边)，而把原来不属于 M 的边加到匹配 M_S 中去(细边改成粗边)，变化后的匹配 M_S 恰好比原匹配 M 多一条边(粗边比细边多一条)。拿图 7-2(a)中的图 G 以及由粗线组成的匹配来说，图 7-2(b)中已经找出了一条可增广轨，因此就可以按刚才所述的“粗、细边对换”的办法把原来的匹配改成一个多包含一条边的匹配了(见图 7-3)。

但是我们必须清醒地认识到，用上述计算方法得到的匹配 M_S，只不过是具有“用在可增广轨上粗细边对换的办法不能再改进了”这样的特点，而这个特点不一定能保证它是最大匹配。判断一个匹配是否是最大匹配的标准只有一条：

若匹配 M 中不存在可增广轨，那末 M 一定是最大匹配。

那么如何在匹配中找可增广轨呢？曾有人提出这样一种方法：

对于图 G 的一个给定的匹配 M 来说，首先看看有几个未盖点，如果没有未盖点或者只有一个未盖点，就不用找了，肯定不会有可增广轨，即 M 是最大匹配。如果未盖点超过一个，就任意取定一个未盖点 V_i，然后从 V_i 出发，用回溯法搜索以 V_i 为首的交错轨。如果搜索了所有可能情况后还未发现一条以另一个未盖点为尾的交错轨，则再取一个未盖点 V_k 进行回溯搜索……，直至找出一条可增广轨或者对于所有未盖点来说，都不存在一条以它为一端的可增广轨为止。

很显然，如果遇到一个比较复杂的图，这种方法是很难用的。这种方法的主要缺点，在于搜索时太不系统，有些边和顶点在回溯时往往反复走过好几次。为了克服搜索不够系统这一缺点，并且设计一种系统搜索可增广轨的方法，我们还需要介绍几个概念：

设 M 是图 G 的一个匹配，V_i 是取定的未盖点，如果存在着 V_i 与另一顶点 V_j 的交错路，就称由 V_i 交错可达 V_j(亦可认为 V_i 交错可达 V_i 本身)

以图 7-3(b)为例，如果我们取定了未盖点 V_4，那么因为存在着交替轨

$$P=\{V_4,V_3,V_7,V_6\}$$

因此由 V_4 交替可达 V_6，当然，由 V_4 也交替可达 V_7，因为 P 中从 V_4 到 V_7 这一段也是交错轨，不难看出，由 V_4 还交错可达 V_5 与 V_{10}。

有了交错可达的定义，我们可以看出，如果发现一个未盖点 V_j，而由 V_i 又交错可达 V_j，我们就找到了一条连接 V_i 与 V_j 的交错轨，也是就是一条可增广轨了。因此，要想找以 V_i 为一端的可增广轨，可以用下面的方法：设法把由 V_i 交错可达的顶点一个一个地找出来，每找到一个，就检查一下它是不是未盖点，是的话，可增广轨就找到了。如果已经把所有由 V_i 交错可达的顶点都找出来，而其中没有一个是未盖点，就可以肯定以 V_i 为一端的可增广轨一定不存在了。

为了把由 V_i 交错可达的顶点都找出来，还要用到一个交错树的概念：

设 M 是图 $G=[V,E]$的一个取定的匹配，T 是图 G 的一个子图，如果 T 具有下述性质：

(1) T 是一个树；

(2) T 中存在一个顶点 V_i，它是未盖点；

(3) 对于 T 的任意一个不同于 V_i 的顶点来说，T 中连接 V_i 与 V_j 的唯一轨是交错轨。

那么称 T 是一个以 V_i 为根的交错树。

以图 7-4(a)中的图 G 为例(粗边代表一个匹配)，图 7-4(b)中的子图 T 就是一个以 V_1 为根的交错树。

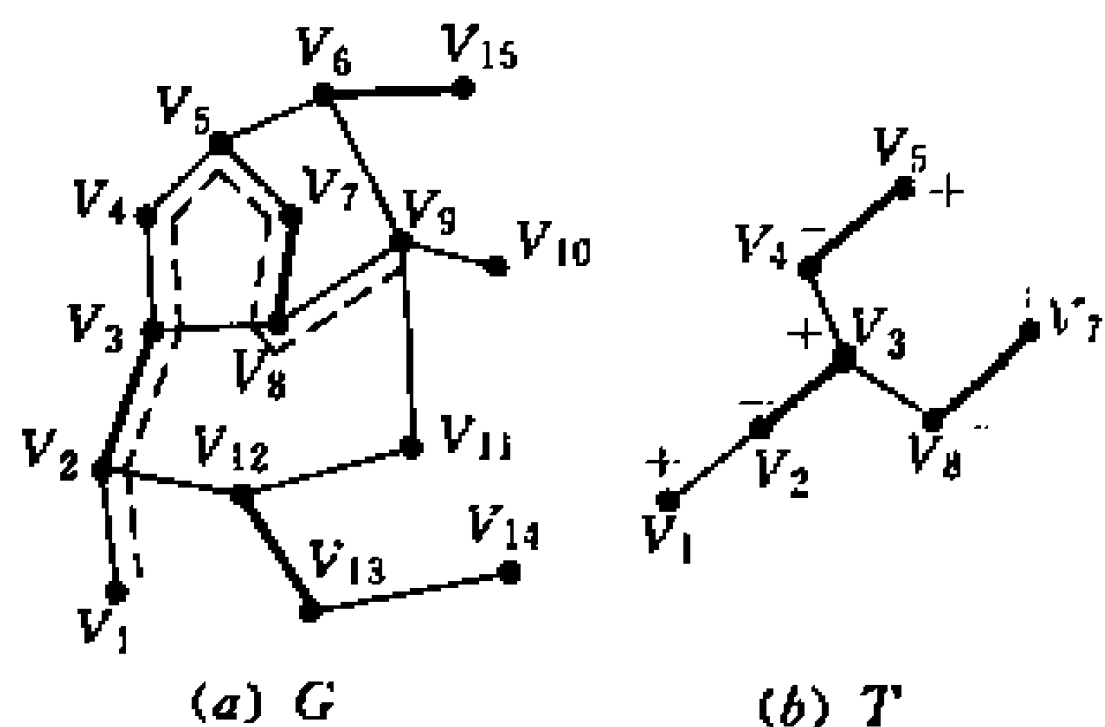


图 7-4

由交错树的定义(3)，不难看出：交错树上的顶点 V_j 都是由 V_i 交错可达的顶点。另外，要具体地把连接 V_j 与根 V_i 的交错轨找出来也不难，因为在 T 中，连接 V_i 与 V_j 的轨是唯一的，而由交错树的定义(3)，这条轨就是要找的交错轨。

因此，想要把由 V_i 交错可达的顶点都找出来，可以采用不断扩大交错树的办法。图 G 开始时，令交错树 T 只含一个顶点，就是取定的未盖点 V_i，V_i 是 T 的根，以后逐步扩大 T，每扩大一次，T 中新增加的顶点就是新找到的由 V_i 交错可达的顶点。

为了讲怎样扩大一个交错树，还要介绍一个有关顶点分类的概念。

T 的顶点可以分成两类：

一类叫外点(即图 7-4(b)中标'+'的顶点)。如果 V_j 是外点，连接根 V_i 与 V_j 的交错路 P 一定含偶数条边，且 P 上与 V_j 关联的边一定属于匹配 M。

一类叫内点(即图 7-4(b)中标'−'的顶点)。如果 V_j 是内点，连接根 V_i 与 V_j 的交错路 P 一定含奇数条边，且 P 上与 V_j 关联的边不属于匹配 M。

有了以上的准备，就可以讲扩大一个以 V_i 为根的交错树的办法了。就是：

看看图 G 中有没有与交错树 T 的外点关联而不属于 T 的边 E，如果有，就再看看 E 的 V_k 另一个端点 V_k 是不是属于 T，如果 V_k 不属于 T，就可以把 E 和 V_k 都加到 T 中去，使 T 扩大。具体扩大的时候，还可以分两种情况：

1. V_k 是未盖点，这时，把 E 与 V_k 加到 T 中去后，T 中连接根 V_i 与 V_k 的交错路是一条可增广轨(见图 7-5(a))；

2. V_k 不是未盖点，也就是说，有一条属于匹配 M 的边 E' 与 V_k 关联，这时，在把 E 与加到 T 中去，接着还可以把 E' 以及它的端点加到 T 中去，因为很显然从 V_i 边也交错可达 E' 的另一个端点 V_1。另外，还容易看出，V_k 应该是内点，而 V_1 是外点(见图 7-5(b))。

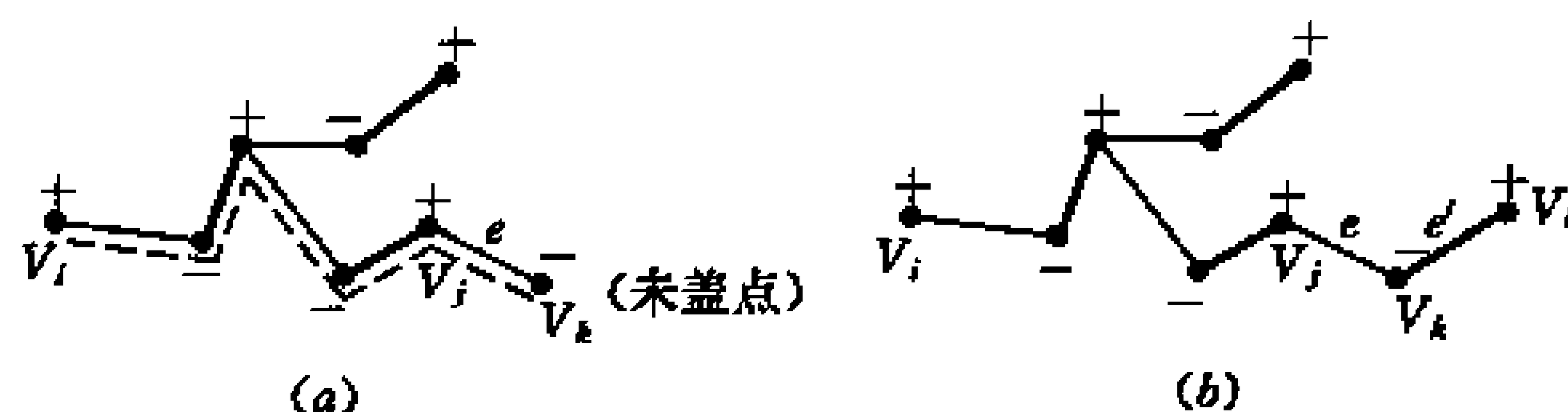


图 7-5

针对图 7-4(a)中的图 G，图 7-6(a)，(b)，(c)，(d)，(e)给出了怎样用上述方法扩展交错树的具体过程：

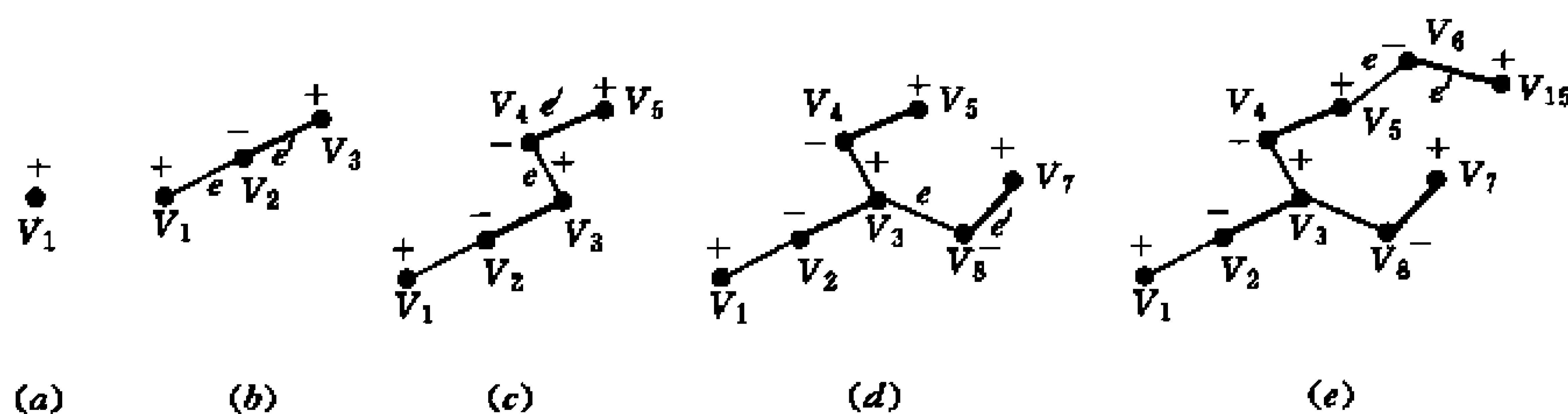


图 7-6

对于图 7-6(e)中的交错树，不能再用上述方法扩大了，因为找不到一条不属于 T 的边 E，这条边与 T 的某外点关联，且 E 的另一端点 V_k 也不属于 T。但是能不能就此断定

以 V_1 为一端的可增广轨一定不存在呢？还不能。事实上，以 V_1 为一端的可增广轨是存在的。图 7-4(*a*)用虚线标出的($V_1,V_2,V_3,V_4,V_5,V_7,V_8,V_9$) 就是一条连接 V_1 与 V_9 的可增广轨。

在求任意图的最大匹配的过程中，上述情况经常出现。为了求得可增广轨，还需要一个重要的概念——带花树。

什么是带花树，就是如果发现了一条不属于交错树 T 的边 E，E 的两个端点都是 T 的外点，那么把 E 加到 T 中去得到的图叫做带花树。

带花树的特点是，它恰好有一个圈，今后我们就把这唯一的圈叫做花。请大家记住，花就是一个圈，圈内包含奇数条边。

例如图 7-6(*e*)中，连接 T 的两个顶点 V_5 与 V_7，这条边原不属于 T，现在把 E 加到 T 中去，只不过是使 T 增加了一条边，没有扩大由 V_i 交错可达的顶点的范围，反而使得 T 不再是树了。这样做，又有什么用呢？我们把这个问题留在"7.5 求任意图的最大匹配"一节中解决。

7.2 求二分图的最大匹配

一、问题及其分析

求二分图的最大匹配的算法是以"6.1 求网络的最大流"一节中所述的方法为基础的。

1. 从二分图 G 出发构造一个网络 $\overline{G}$：

(1) 增加一个源点 S 和一个源点 T；

(2) 从 S 向 X 的每一个顶点都画一条有向弧，从 Y 的每一个顶点都向 T 画一条有向弧；

(3) 原来 G 中的边都改成有向弧，方向是从 X 顶点指向 Y 的顶点；

(4) 令所有弧的容量都等于 1(见图 7-7)。

2. 求网络 $\overline{G}$ 的最大流 F。

3. 从 X 的顶点指向 Y 的顶点的弧集中，弧流量为 1 的弧对应二分图的匹配边，最大流值 F 对应二分图的最大匹配的边数。

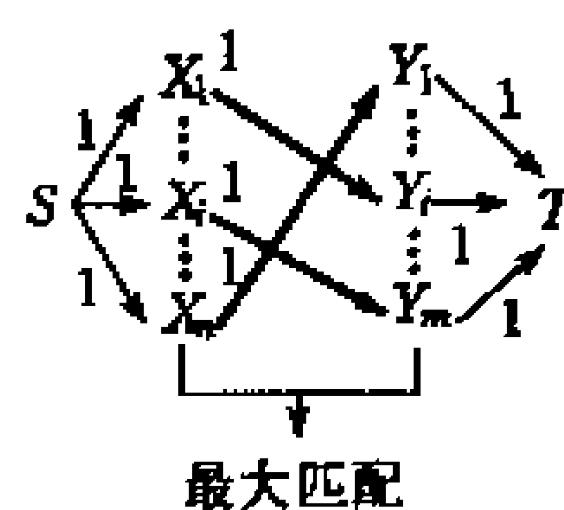


图 7-7

当我们对二分图 G 进行"覆盖"的分析后，会引出一个有趣的结论：

最大匹配的边数等于 G 的覆盖数 $\alpha(G)$。

二、程序

下面给出程序题解。

```
program maxmatch;
{ 最大匹配 }
```

```
const
  MaxN = 30;
type
  nodetype = record { 可增广轨的顶点类型 ：标号、检查标志 }
    l, p: integer;
  end;
  arctype = record { 网顶点类型 ：容量、流量 }
    c, f: integer;
  end;
  gtype = array [1..maxn, 1..maxn] of arctype;   { 网类型 }
  ltype = array [0..maxn] of nodetype;            { 可增广轨类型 }
var
  lt: ltype; { 可增广轨 }
  g: gtype; { 网 }
  n, s, t, n1, n2: integer; { 顶点数、源点、汇点、|x|、|y| }
procedure readgraph;
var
  s: string;
  i, j: integer;
  f: text;
begin
  write('Graph filename > ');        { 读入文件名并与文件变量连接 }
  readln(s);
  assign(f, s);
  reset(f);                          { 读准备 }
  readln(f, n1, n2);                 { 读入|x|和|y| }
  n := n1+n2+2;                      { 计算 N 网的顶点数 }
  fillchar(g, sizeof(g), 0);         { N 网和可增广轨初始化 }
  fillchar(lt, sizeof(lt), 0);       { 从 s 向 x 的每一顶点引出容量为 1 的有向弧 }
  for i := 2 to n1+1 do g[1, i].c := 1;
  { 从 y 的每一顶点向 t 引出容量为 1 的有向弧 }
  for i := 2+n1 to n-1 do g[i, n].c := 1;
  { 原二分图的边改为从 x 的顶点指向 y 的顶点的有向弧 }
  for i := 1 to n1 do
    for j := 1 to n2 do
      read(f, g[1+i, 1+n1+j].c);
  close(f);
end;
function check: integer; { 返回一个已标号而未检查的顶点序号 }
var
  i: integer;
begin
  i := 1;
  while (i<=n) and not ((lt[i].l<>0) and (lt[i].p=0)) do inc(i);
  if i>n then
    check := 0
  else
    check := i;
```

```
end;
function ford(var a: integer): boolean;
{ 若 s 顶点至 t 顶点无可增广轨返回 true,否则返回 false 和可增广轨的改进量 a }
var
  i, j, m, x: integer;
begin
  ford := True;
  fillchar(lt, sizeof(lt), 0); { 撤消增广轨 }
  lt[s].l := s;
  repeat
    i := check; { 寻找一个已标号而未检查的顶点序号 i }
    if i = 0 then Exit; { 若该顶点不存在,则退出过程,返回 true }
    for j := 1 to n do
      if (lt[j].l=0) and ((g[i, j].c<>0) or (g[j, i].c<>0)) then
{ 寻找所有与顶点 i 连接且未标号的顶点 j,弧<i,j>置前向弧或后向弧标志 }
      begin
        if (g[i, j].f<g[i, j].c) then lt[j].l := i;
        if (g[j, i].f>0) then lt[j].l := -i;
      end;
    lt[i].p := 1; { 顶点 i 的所有相关弧分析完,置顶点 i 检查标志 }
  until (lt[t].l<>0); { 循环,直至 t 顶点标号为止 }
  m := t; { 从顶点 t 倒推,改进量初始化 }
  a := MaxInt;
  repeat { 求 a = min[所有前向弧的 C(i,j)-F(i,j),所有前向弧的 F(i,j)] }
    j := m;
    m := abs(lt[j].l);
    if lt[j].l<0 then x := g[j, m].f;
    if lt[j].l>0 then x := g[m, j].c-g[m, j].f;
    if a>x then a := x;
  until m = s;          { 直至倒推至顶点 s 为止 }
  ford := False; { 返回可增广轨存在标志 false }
end;
procedure fulkerson(a: integer);
var
  m, j: integer;
begin
  m := t; { 从 t 顶点出发,沿增广轨修正流量 }
  repeat
    j := m;
    m := abs(lt[j].l);
    if lt[j].l<0 then g[j, m].f := g[j, m].f-a;
    if lt[j].l>0 then g[m, j].f := g[m, j].f+a;
  until m = s;
end;
procedure answer; { 打印从 x 的顶点指向 y 的顶点的弧集中流量为 1 的匹配边 }
var
  i, j: integer;
begin
```

```
  for i := 2 to n-1 do
    for j := 2 to n-1 do
      if g[i, j].f=1 then writeln(i-1, '------', j-n1-1);
end;

procedure proceed;
var
  del: integer;
  success: boolean;
begin
      s := 1; { 设置源点和汇点 }
      t := n;
      repeat
         success := ford(del);        { 寻找可增广轨和可改进量 del }
         if success then          { 若可增广轨不存在,打印最大匹配 }
           answer
        else
           fulkerson(del);             { 否则沿增广轨修正流量 }
      until success;
end;

begin
  readgraph;  { 读入二分图,构造 N 网 }
  proceed;    { 通过求 N 网的最大流得出二分图的最大匹配 }
end.
```

7.3 求二分图的完备匹配

一、分工问题和求二分图的完备匹配

某公司有工作人员 $X_1, X_2, \cdots, X_m$,他们去做工作 $Y_1, Y_2, \cdots, Y_n$,每人适合做其中的一项或几项工作,问能否每人都分配一项合适的工作。

这个问题的数学模型是:构造一个二分图 G,顶点划分为两个部分:一个是工作人员集合 $X=\{X_1, X_2, \cdots, X_m\}$,一个是工作集合 $Y=\{Y_1, Y_2, \cdots, Y_n\}$,当且仅当 X_i 适合干工作 Y_j 时,X_i 与 Y_j 之间连一条边。问是否能从 G 中得出一个不含未盖点的匹配,这种将 G 的每一个顶点盖住的匹配称为二分图的完备匹配。

可能存在完备匹配的图是否仅限于二分图呢?答案是否定的。数学家在做了大量的图论分析后得出:

(1) G 是 $K(K>0)$ 次正则二分图;

(2) G 是无桥的三次正则图;

(3) G 在去除任意一个顶点子集 S 后,其子图中含顶点数为奇数的连通分支个数不大于 $|S|$。

具有上述 3 个特征的图肯定有完备匹配。特别是特征(3),是完备匹配的充分必要条件:含特征(3)的图一定有完备匹配;有完备匹配的图也一定含有特征(3)。

下面介绍一种求二分图最佳匹配的算法——匈牙利算法。

变量说明：

S——外点集合，初始时为交错树的根；

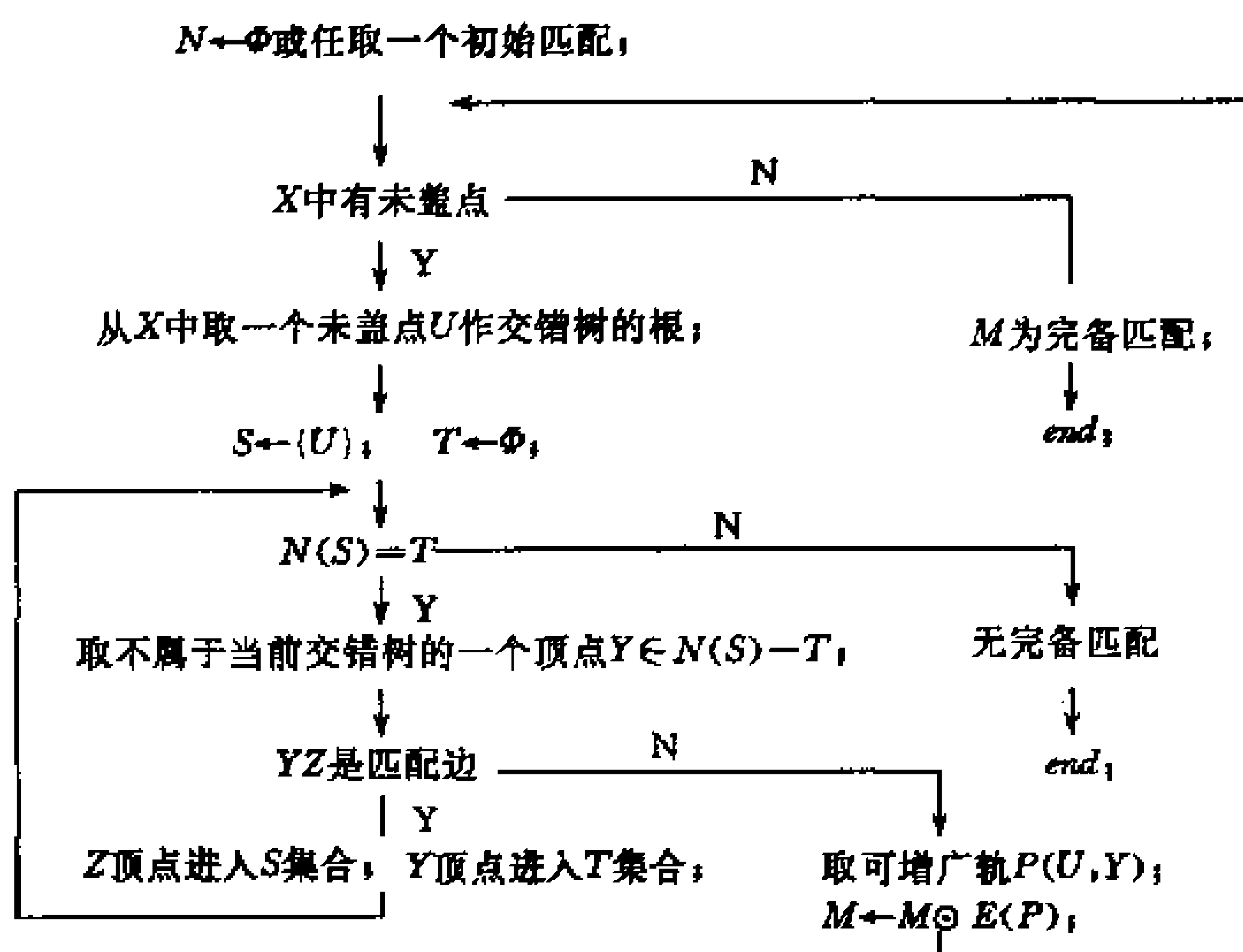
T——内点集合，初始时为空；

$N(S)$——与 S 中的顶点相邻的顶点集合。显然，若 $N(S)=T$，表明与任一外点相连的端点都在内点集合里，找不出一条不属于交错树的边，图 G 无完备匹配；

M——匹配边集合；

$E(P)$——可增广轨 P 上的边。

$M\odot E(P)$——进行对称差运算的结果，为可增广轨 P 上的匹配边与非匹配边对换，P 轨外的匹配边不变，使得 M 改进成一个多包含一条边的匹配。



这个算法的要点是把初始匹配通过可增广轨逐次增广，以至得到最大匹配。然后根据有无未盖点来判定这个最大匹配是否为完备匹配。

例如求图 7-8(a)中的最大匹配，设初始匹配 $M=\{X_2Y_2,X_3Y_3,X_5Y_5\}$。

以未盖点 X_1 为根，生成交错树，结果得到可增广轨 $X_1Y_2X_2Y_1$(见图 7-8(b))。

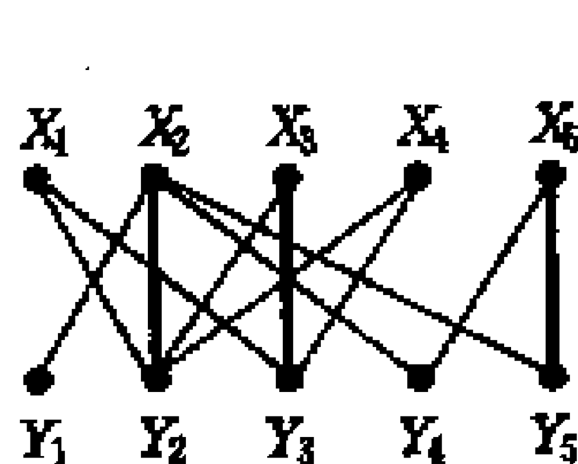


图 7-8(a)

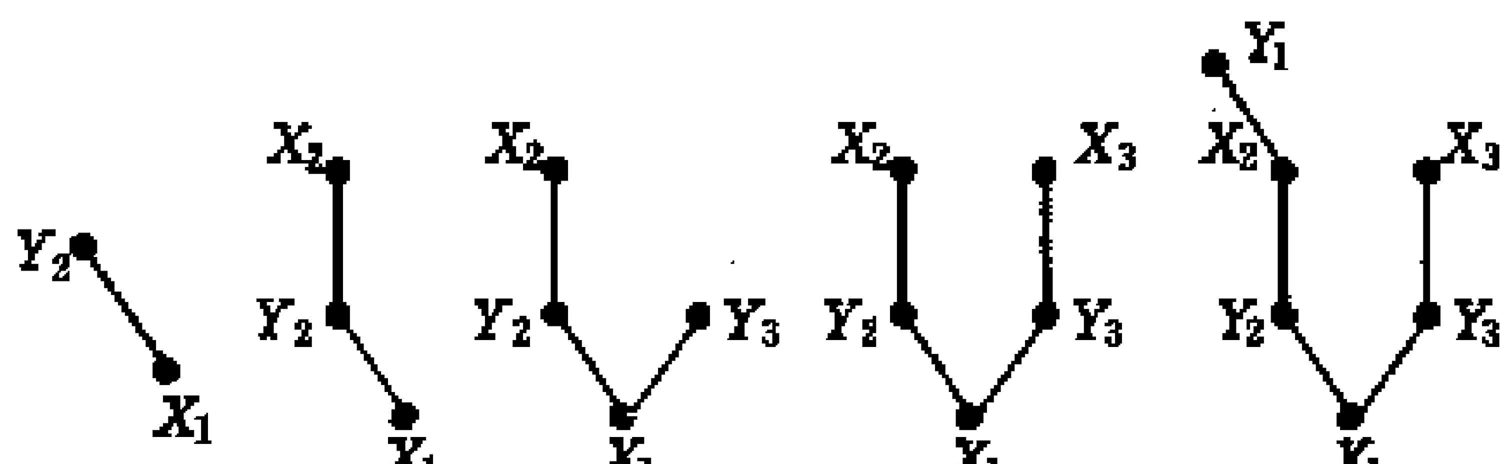


图 7-8(b)

通过可增广轨把 M 增广成 $M_1=\{X_1Y_2,X_2Y_1,X_3Y_3,X_5Y_5\}$(见图 7-8($c$))。

以未盖点 X_4 为根，生成交错树（见图 7-8(*d*)）。

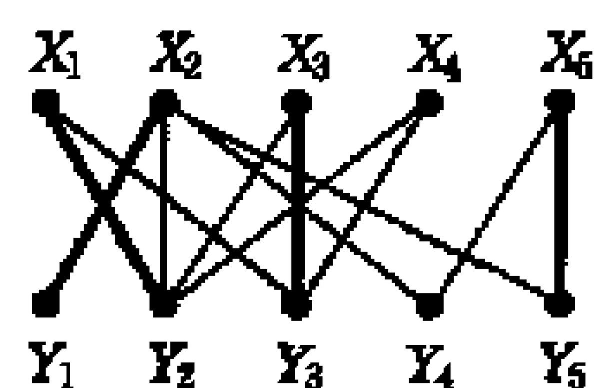


图 7-8(*c*)

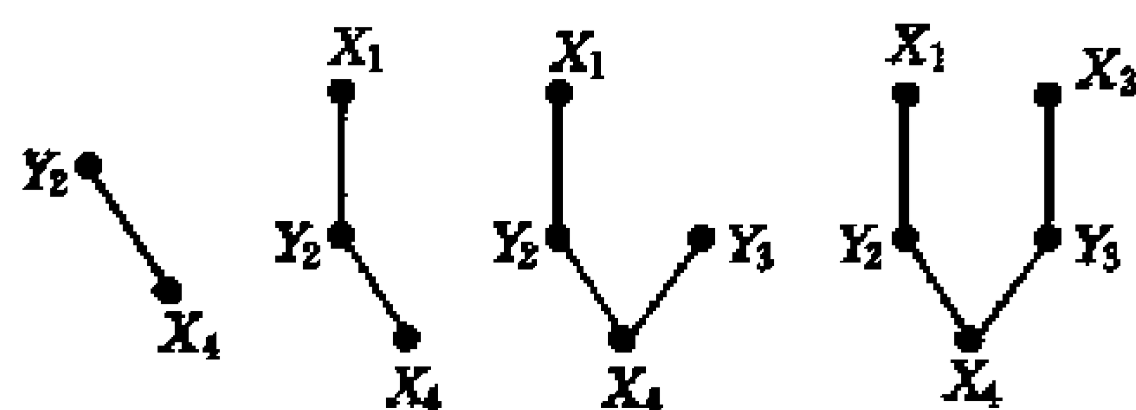


图 7-8(*d*)

结果未得到可增广轨，可见 M_1 是最大匹配，无完备匹配。

二、求二分图完备匹配的程序

```
program bestmatch;
{ 完备匹配 }
const
  maxn = 30;
type
  graphtype = array [1..maxn, 1..maxn] of integer; { 图类型 }
  settype = set of 1..maxn;
var
  g: GraphType;              { 二分图 }
  m, n: integer;             { |y|和|x| }
  s, t, cx, cy: SetType;     { 外点集合,内点集合,已匹配的 x 集合,已匹配的 y 集合 }
  found: boolean;            { 完备匹配存在标志 }
procedure readgraph;
var
  i, j: integer;
  st: string;
  f: text;
begin
  write('Graph filename = '); { 读入文件名并与文件变量连接 }
  readln(st);
  assign(f, st);
  reset(f);                  { 读准备 }
  readln(f, n, m);           { 读入|x|和|y| }
  for i := 1 to n do         { 读入二分图 }
for j := 1 to m do
      read(f, g[i, j]);
  close(f);
    end;
function path(x: integer): boolean;
{ 若扩展出以 x 集合中的未盖点 x 为根的交错树,则返回 true;否则返回 false }
var
  i, j: integer;
begin
```

```
  path := false;
  for i := 1 to m do
    if not (i in t) and (g[x, i]<>0) then
{ 若i不属内点且x至i有边 }
    begin
      t := t+[i]; { 则i进入内点集合,并分析: }
      if not (i in cy) then
{ 若i是y集合中的未盖点, 则<x,i>设为匹配边,i进入已匹配的y集合, 返回true }
      begin
        g[x, i] := -g[x, i];
        cy := cy+[i];
        path := true;
        exit;
      end;
      j := 1;
      while (j<=m) and not (j in s) and (g[j, i]>=0) do inc(j);
      if j<m then { 若<j,i>是匹配边,则j进入外点集合 }
      begin
        s := s+[j];
        if path(j) then              { 若从j出发递归扩展出交错树 }
        begin     { 则可增广轨的匹配边与非匹配边对换,并返回true }
          g[x, i] := -g[x, i];
          g[j, i] := -g[j, i];
          path := true;
          exit;
        end;
      end;
    end;
end;

procedure main;
var
  u, i, j: integer;
begin
  found := true;
  u := 1;
  cx := []; { 已匹配的x集合和y集合设空 }
  cy := [];
  while u<=n do
  begin
    s := [u]; { 取u作外点,内点集合设空 }
    t := [];
    if not (u in cx) then { 若u是x集合中的未盖点 }
    begin
      if not path(u) then { 若扩展不出以u为根的交错树,则无完备匹配 }
      begin
        writeln('No answer.');
        found := false;
        exit;
      end
```

```
      else cx := cx+[u]; { 扩展出以 u 为根的交错树,u 进入已匹配的 x 集合 }
      u := 0;
    end;
    inc(u); { 从 x 集合中取下一个未盖点 u 作交错树根 }
  end;
end;

procedure print; { 打印完备匹配 }
var
  i, j: integer;
begin
  for i := 1 to n do
    for j := 1 to m do
      if g[i, j]<0 then
        writeln(i, '————', j);
end;

begin
  readgraph;                { 读入二分图 }
  main;                     { 求完备匹配 }
  if found then print;      { 若存在完备匹配,则打印结果 }
end.
```

7.4 求二分图的最佳匹配

一、问题及其算法

在 7.3 节例举的分工问题中,工作人员可以做的各项工作,效益未必一致,我们需要制定一个分工方案,使公司的总效益最大,这就是所谓最佳分配问题。它的数学模型如下:

G 是加权完全二分图,顶点划分为两个集合 X,Y; $X=\{X_1,X_2,\cdots,X_n\}$, $Y=\{Y_1,Y_2,\cdots,Y_m\}$, $W(X_i,Y_s)\geqslant 0$ 是工作人员 X_i 做 Y_s 工作时的效益($1\leqslant i\leqslant n,1\leqslant s\leqslant m$),求权最大的完备匹配,这种完备匹配称为最佳匹配。

若用穷举法,需要对 $n!$ 个完备匹配进行比较,当 n 较大时,计算量过大,是不可能很快得到结果的。本节给出一种有效算法,为此,我们首先引入一些概念,作为本节中心算法的基础。

1. 二分图 G 的可行顶标 $L(u)$

对于 X 集合中的任一顶点 x 和 Y 集合中的任一顶点 y,满足条件

$$L(x)+L(y)\geqslant W(xy)$$

则称 $L(u)$ 是 G 的可行顶标。这个可行顶标是存在的,例如初始时可设

$$L(x)=\max_{y\in Y}W(xy) \qquad x\in X$$
$$L(y)=0 \qquad y\in Y$$

2. 相等子图 G_L

由边集 $E_L=\{xy \mid xy\in E$ 的边集 E, $L(x)+L(y)=W(xy)\}$ 组成的 G 的生成子图,称作相等子图,记为 G_L。

为什么我们要给出上述两个概念，原因是欲求二分图 G 的最佳匹配，只需用匈牙利算法求取相等子图 G_L 的完备匹配，即为 G 的最佳匹配。但问题是，当 G_L 中无完备匹配时怎么办，下述算法给出修改顶标的一个方法，使新的相等子图的最大匹配逐渐扩大，最后出现相等子图的完备匹配。

变量说明：

L——可行顶标集合；

I——修改后的顶标集合；

G_L——修改顶标后的相等子图；

A_L——顶标的改进量；

S——G_L 的匹配中，外点的集合，初始时为交错树的根；

T——G_L 的匹配中，内点的集合，初始时为空；

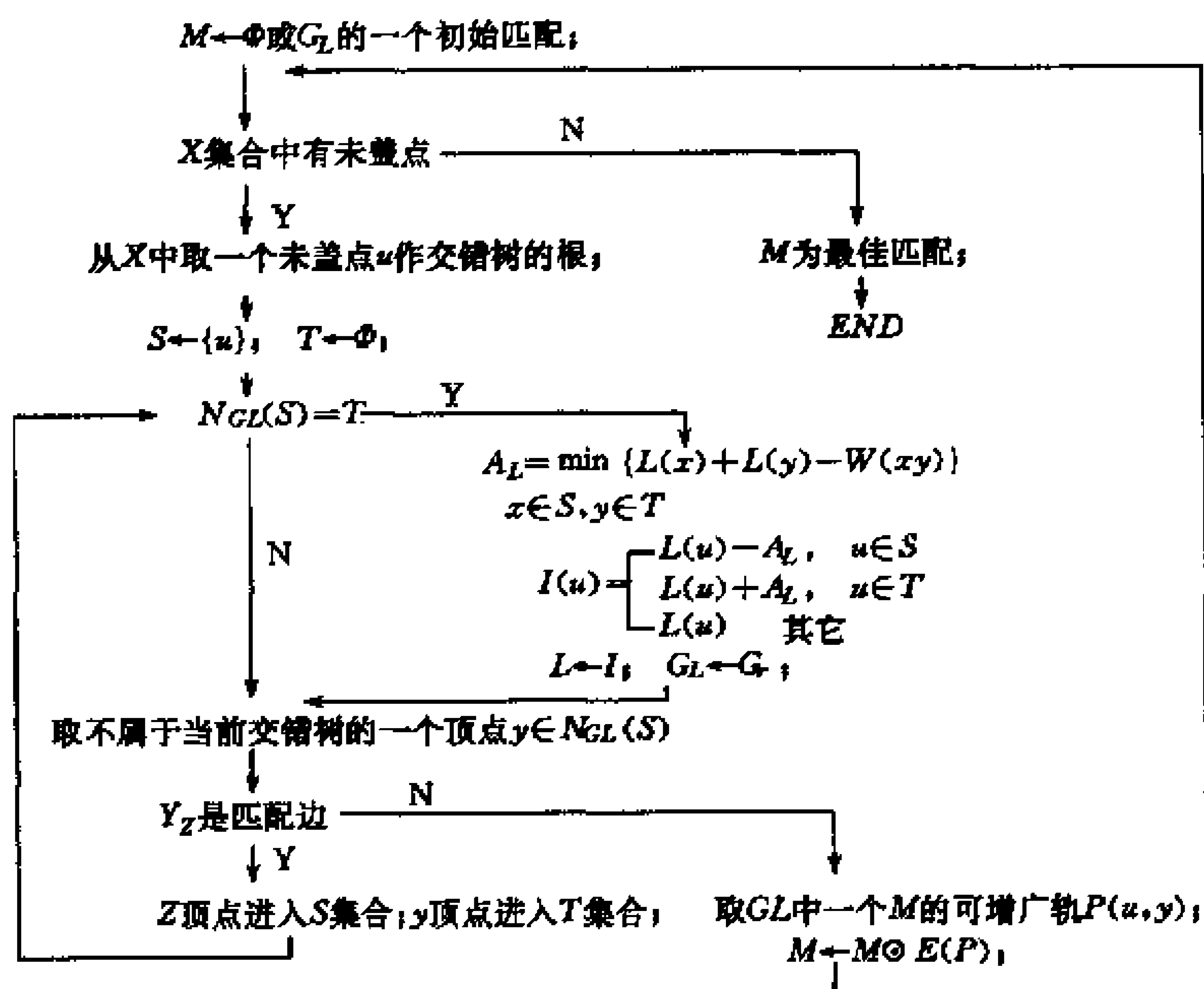
$N_{GL}(S)$——G_L 中与 S 相邻的顶点集合。显然，若 $N_{GL}(S)=T$，G_L 无完备匹配。M、$E(P)$ 和对称差运算 $M\odot E(P)$ 的意义如完备匹配算法。

下面给出求二分图的最佳匹配的算法流程：

选定初始的可行项标 L：

$$L(x)=\max_{y\in Y}W(xy),\qquad x\in X$$
$$L(y)=0\qquad y\in Y$$

确定 G_L：



例：已知 $K_{5,5}$ 的权矩阵为

$$G = \begin{array}{c} \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \end{array} \begin{array}{c} \begin{array}{ccccc} Y_1 & Y_2 & Y_3 & Y_4 & Y_5 \end{array} \\ \begin{bmatrix} 3 & 5 & 5 & 4 & 1 \\ 2 & 2 & 0 & 2 & 2 \\ 2 & 4 & 4 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 2 & 1 & 3 & 3 \end{bmatrix} \end{array}$$

求最佳匹配，其中 $K_{5,5}$ 的顶点划分为 $X=\{X_i\}, Y=\{Y_i\}, i=1..5$。

我们利用上述算法求解：

1. 取初始的可行顶标 $L(u)$ 为：

$L(Y_i)=0 \quad i=1,2,\cdots,5$

$L(X_1)=\max\{3,5,5,4,1\}=5$　　$L(X_4)=\{0,1,1,0,0\}=1$

$L(X_2)=\max\{2,2,0,2,2\}=2$　　$L(X_5)=\{1,2,1,3,3\}=3$

$L(X_3)=\max\{2,4,4,1,0\}=4$

2. 确定 G_L 及其上的一个初始匹配 $M=\{X_1Y_2, X_2Y_1, X_3Y_3, X_5Y_5\}$（见图 7-9(*a*)）。

3. 以 $u=X_4$ 为根，求交错树，得 $S=\{X_4, X_3, X_1\}, T=\{Y_3, Y_2\}$。$N_{GL}(S)=T$，于是

$$A_L = \min_{x \in s, y \notin t} \{L(x) + L(y) - W(xy)\} = 1$$

修改顶标：

$I(X_1)=4$;　　$I(Y_1)=0$;

$I(X_2)=2$;　　$I(Y_2)=1$;

$I(X_3)=3$;　　$I(Y_3)=1$;

$I(X_4)=0$;　　$I(Y_4)=0$;

$I(X_5)=3$;　　$I(Y_5)=0$

4. 用修改后的顶标 I 得 G_I 及其上面的一个完备匹配。

图 7-9(*b*)中粗实线给出了一个最佳匹配，其最大权是 2+4+1+4+3=14。

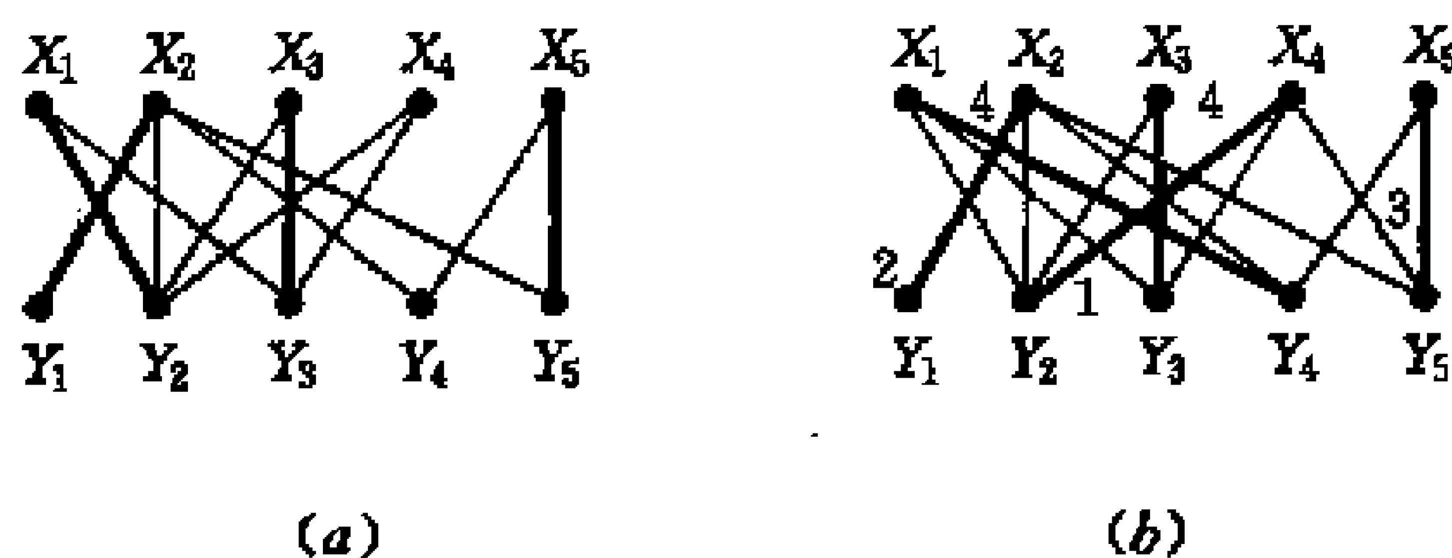


图 7-9

由上可以看出：$A_L>0$；修改后的顶标仍是可行顶标；G_I 中仍含 G_L 中的匹配 M，G_I 中至少会出现不属于 M 的一条边，所以会造成 M 的逐渐增广。

二、求二分图最佳匹配的程序

```
program kuhn_munkras_algorithm;
```

```
{ 最佳匹配程序 }
 const
   maxn          =  30;
 type
   gtype         =  array [1..maxn,1..maxn] of integer;  { 二分图类型 }
   ltype         =  array [1..maxn] of integer;           { 顶标类型 }
   settype       =  set of 1..maxn;
 var
   n,m           :  integer;        { |x|,|y|}
   g,gc          :  gtype;          { 图G矩阵,中间矩阵 }
   l             :  ltype;          { 图G矩阵顶标 }
   f             :  text;           { 文件变量 }
   s,t,cx,cy     :  settype;
   { 已检查的x集合,已检查的y集合,已匹配的x集合,已匹配的y集合 }
 procedure read—graph;
   var
     str:string;
     i,j:integer;
   begin
     write('Graph file = '); { 读入文件名,并与文件变量连接 }
     readln(str);
     assign(f,str);
     reset(f);
     readln(f,n,m);          { 读入|x|和|y| }
     for i:=1 to n do        { 读入 xi 做 yj 工作的效益 }
       for j:=1 to m do read(f,g[i,j]);
     close(f);
   end;
 function path(x:integer):boolean;
 { 求x出发的增广轨。若存在,则增广轨上的匹配边与非匹配边对换 }
   var
     i,j:integer;
   begin
     path:=false;
     for i:=1 to m do
       if not (i in t) and (g[x,i]<>0) { 若i∉已检查的y集合,且(x,i)∈E }
         then begin                  { i进入已检查的y集合 }
                t:=t+[i];
                if not (i in cy)
                  { 若i∉已匹配的y集合,则<x,i>置匹配边标志 }
                  { i进入已匹配的y集合,返回 true }
                  then begin
                         g[x,i]:=-g[x,i];cy:=cy+[i];
                         path:=true;exit
                       end;
                j:=1;
                { 与x相邻的顶点i已匹配,搜索一条已 }
```

```
              { 匹配边<j,i>(j∉已检查的 x 集合) }
              while (j<=m) and not (j in s) and (g[j,i]>=0) do inc(j);
              if j<=m
                 then begin
                        s:=s+[j];   { j 顶点进入已检查的 x 集合 }
                        if path(j)    { 若 j 出发存在增广轨 }
                           then begin
                                  g[x,i]:=-g[x,i];g[j,i]:=-g[j,i];
                                  { 则增广轨的匹配边与非匹配边对换 }
                                  path:=true;exit
                                end
                      end
            end
  end;
procedure kuhn—munkras;
  var
    u,i,j,al:integer;
  begin
    fillchar(l,sizeof(l),0); { 可行顶标初始化 }
    gt:=g;
    for i:=1 to n do             { 求可行顶标 l(xi)=max(w(xi,yj)) }
      for j:=1 to m do          {                    yj∈y            }
        if l[i]<g[i,j] then l[i]:=gt[i,j];
    u:=1;cx:=[];cy:=[]; { 已匹配的 x 顶点集合和 y 顶点集合初始化 }
    for i:=1 to n do     { 求 CL}
      for j:=1 to m do
        if l[i]+l[n+j]=gt[i,j]
           then g[i,j]:=1
           else g[i,j]:=0;
    while u<=n do
      begin
        s:=[u];t:=[];
        if not (u in cx) { 若 u 不属于匹配的 x 顶点集合 }
           then begin
                  if not path(u) { 若找不到从 u 出发的一条可增广轨 }
                     then begin
                            al:=maxint;
                            { 求 aL= min { l(x)+l(y)-w(x,y) }
                                   x∈S,y∉t
                            {                              }
                            for i:=1 to n do
                              for j:=1 to m do
                                if (i in s) and not (j in t)
                                   and (l[i]+l[n+j]-gt[i,j]<al)
                                   then al:=l[i]+l[n+j]-gt[i,j];
                            { 修改顶标,形成GI'}
                            for i:=1 to n do if i in s then l[i]:=l[i]-al;
                            for i:=1 to m do if i in t then l[n+i]:=l[n+ i]+al;
                            for i:=1 to n do { Gl←GI'}
                              for j:=1 to m do
```

```
                                        if l[i]+l[n+j]=gt[i,j]
                                           then g[i,j]:=1
                                           else g[i,j]:=0;
                                   cx:=[];cy:=[]
                                 end { then }
                           else cx:=cx+[u]; { u 进入已匹配的 x 顶点集合 }
                         u:=0
                       end; { then }
              inc(u)       { 从下一顶点开始搜索 }
            end
        end;
      procedure print; { 打印最佳分配 }
        var
          i,j,tot:integer;
        begin
          tot:=0;
          for i:=1 to n do
            for j:=1 to m do
            { 若第 i 个人能做第 j 项工作(工作效益为 gt[i,j]),则打印 }
              if g[i,j]<0
                 then begin
                        writeln(i,'——',j);tot:=tot+gt[i,j]
                      end;
          writeln('Tot = ',tot)
        end;
      begin
        read_graph;   { 读入二分图 }
        kuhn_munkras; { 求最佳匹配 }
        print         { 打印结果 }
      end.
```

7.5 求任意图的最大匹配

一、问题及其分析

在"7.1 匹配的基本概念"一节中曾讲过：在扩展以 V_i 为根的交错树 T 的一个外点 V_j 时，若找不到一条不属于 T 的边(V_j,V_k)(其中 V_k 也不属于 T)，则在 G 中寻找另一条不属于 T 且连接 T 上两个外点的边 E。若这样的边 E 不存在，则断定以 V_i 为一端的可增广轨一定不存在；否则，将 E 加到 T 中，使 T 成为一个带花的树，这个带花的树有什么作用，这是我们在这一节里所要讲的内容。

在出现带花的树时，我们按下述方法将花 B 收缩成一个顶点 V_B，使得图 G 变成另一个图 G_1：将 G 中所有两个端点都属于 B 的边收缩掉，而 G 中仅一个端点属于 B 的边变成 G_1 中以 V_B 为一端的边。

例如图 7-10 中画的就是从 G 收缩成 G_1 的过程。

在把 G 收缩成 G_1 的过程中，G 的子图边就变成 G_1 的子图了。特别是，G 中带花的树 T 恰好收缩成 G_1 的交错树 T_1（见图 7-11）。

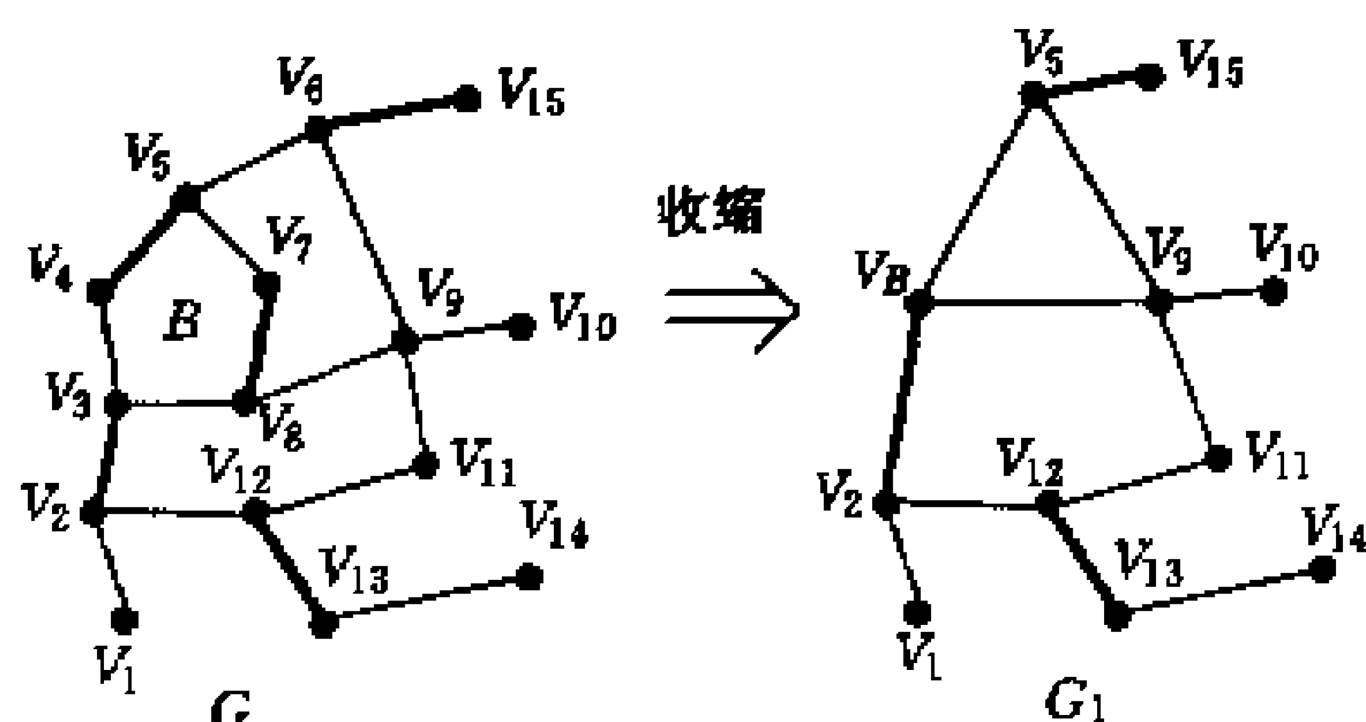


图 7-10

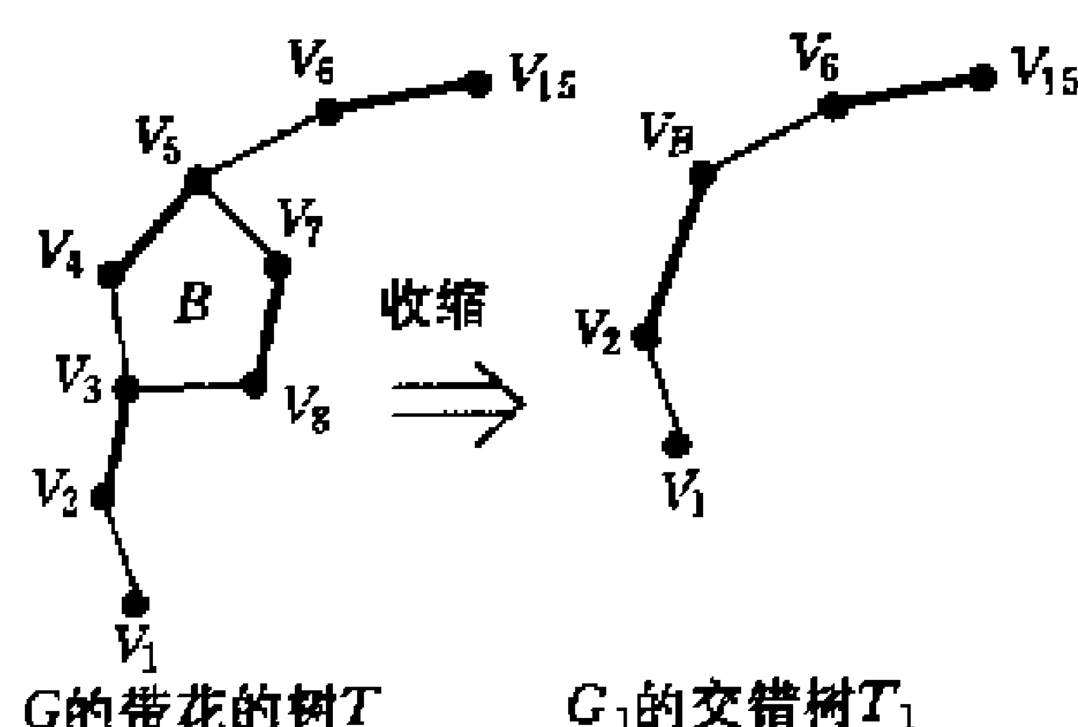


图 7-11

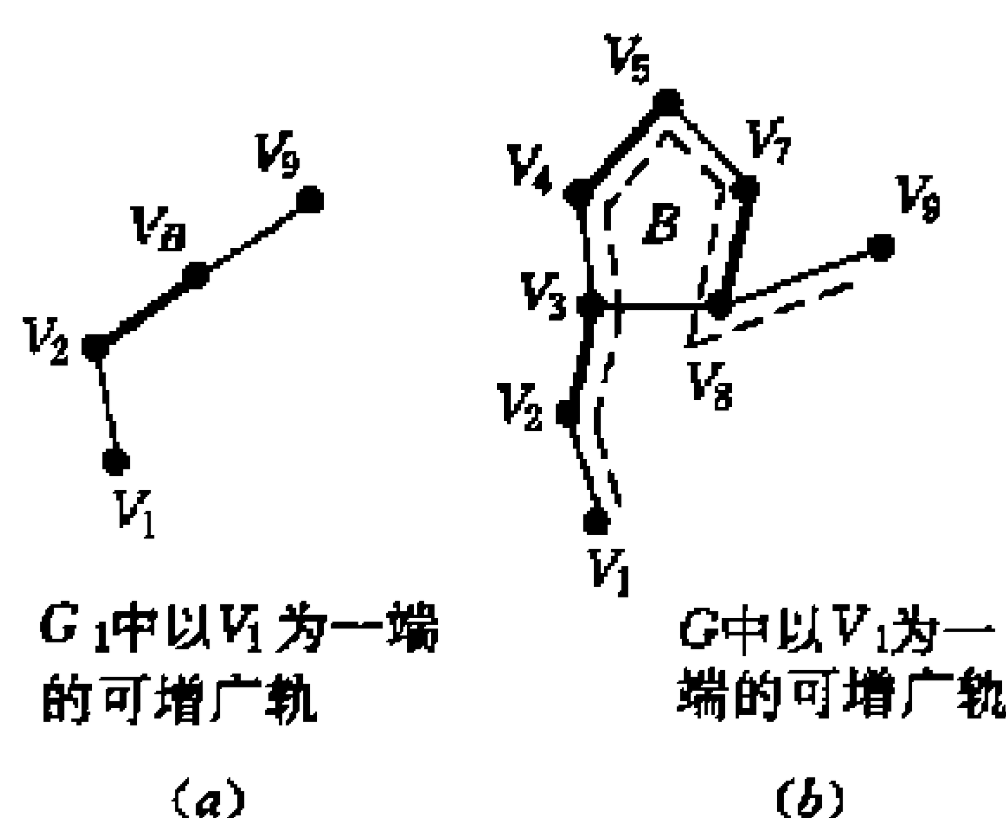


图 7-12

接下去，我们就在 G_1 中找以根为一端的可增广轨，当然不必从头做起，而就在 T_1 的基础上来找，也就是在 G_1 中扩大 T_1。在扩大的过程中，如果找到了可增广轨，这时我们也就找到 G 的可增广轨了。为什么呢？让我们仍旧结合上面的例子讲，从图 7-10 中可以看出，G_1 中有一条连接 V_B 与 V_9 的边，在这条不属于 T_1 的边 (V_B, V_9) 中，V_B 是 T_1 的外点，V_9 是未盖点，我们在 G_1 中就找到了一条可增广轨（见图 7-12(*a*)）。这条轨经过收缩点 V_B，这时应该把 V_B"展开"成花 B，展开后很容易找到原图 G 的一条以 V_1 为一端的可增广轨，也就是图 7-12(*b*)中用虚线标出的那一条轨。

在 G_1 中扩展交错树 T_1 时，如果出现了一条连接 T_1 的一个外点且不属于 T_1 的边 E 时，就扩大 T_1；而如果又发现连接两个外点的边时，就再收缩，即把 G_1 收缩成另一个图 G_2，这时 T_1 就收缩成交错树 T_2。如果在 G_2 中找到了可增广轨，就要两次把收缩点展开成花，来找 G 中的可增广轨。

讲到这里，找以未盖点 V_i 为一端的可增广轨的方法基本上讲完了，总结起来，就是用 7.1 节中所讲的方法和上面所讲的方法不断地扩大交错树 T。当然在遇到带花的树时就应该收缩，在扩大 T 的过程中，如果发现可增广轨，那么我们的目的就达到了。如果一直没有发现可增广轨而又出现无法用 7.1 节和这一节所讲的方法再扩大 T 的情况，则可以断定，一定不存在以 V_i 为一端的可增广轨了。这时应该再取一个不同于 V_i 的未盖点 V_j，然后扩大以 V_j 为根的交错树。当然这样做的结果或者是找到了可增广轨，或者是断定以 V_j 为一端的可增广轨不存在。然后再取一个没有考虑过的未盖点……，如果所有未盖点都考虑了一遍，每次都是出现无法再扩展的情况，那么就可以肯定不会有可增广轨了，从而断定现有的匹配是最大匹配。

例如，我们将图 7-10 作为计算例子。取这个图中的粗边作为初始匹配 M_1。对于 M_1 来

说有未盖点 V_1,V_9,V_{10}, V_{11}和 V_{14},我们用扩展以未盖点 V_1 为根的交错树的办法,找到了一条可增广轨,即图 7-13 中用虚线标出的那一条。

在这条可增广轨上,可用粗细边对换的办法,把匹配 M_1 改进成图 7-14 中用粗边表示的匹配 M_2。对于匹配 M_2 来说,还有 3 个未盖点 V_{10},V_{11}和 V_{14},现在就取定 V_{10},然后来扩大以 V_{10}为根的交错树 T(见图 7-15(a))。

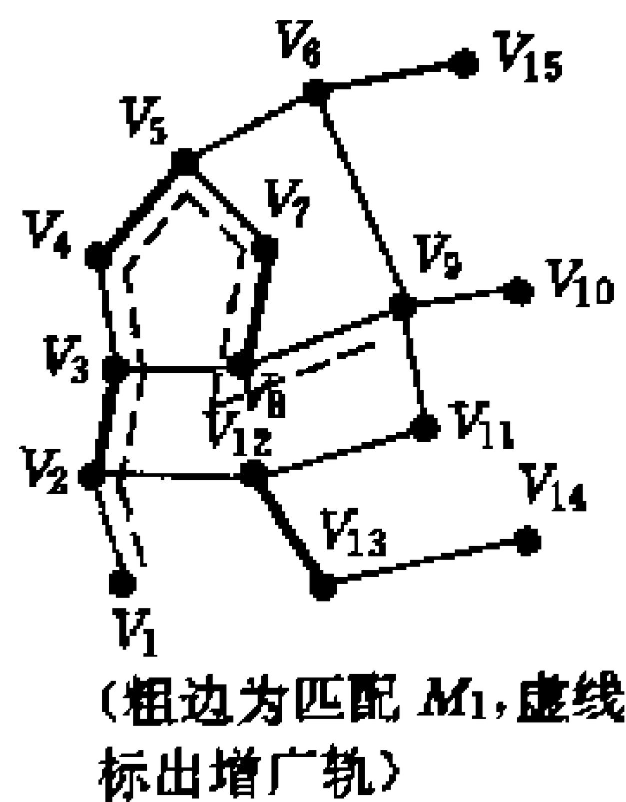


图 7-13

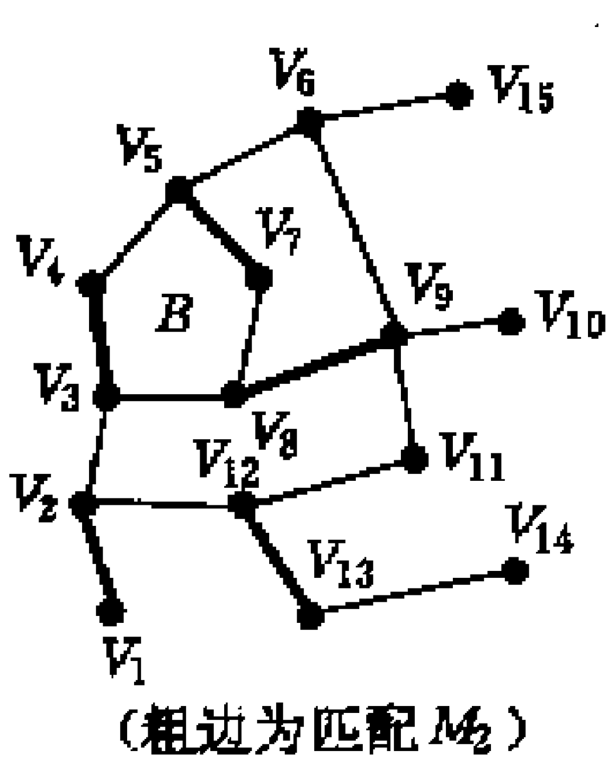


图 7-14

由于 T 无法扩展出可增广轨,因此只得将不属于 T 且连接 T 的两个外点的边(V_4,V_5)加到 T 中去,使 T 或为带花的树(见图 7-15(b))。然后把花收缩成一个顶点,从而把带花的树 T 又变成交错树 T_1 了。(见图 7-15(c))

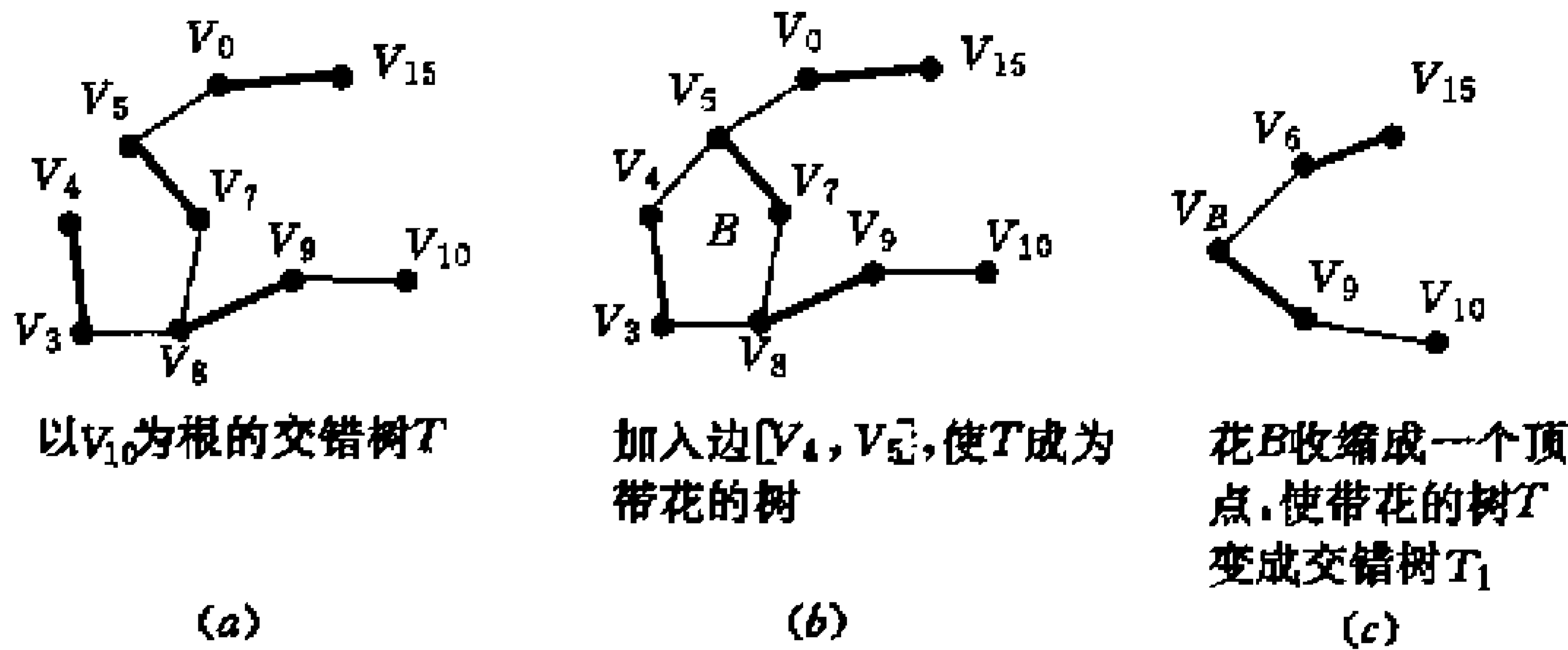


图 7-15

当然,这时我们也应把图 G 中的圈 B 收缩成一个顶点 V_B,使 G 变成另一个图 G_1。G_1 这里就不画了。不难看出,收缩后的边(V_2,V_3)与 T_1 的外点 V_B 关联,且这条边的另一个端点 V_2 不属于 T_1,因此 T_1 又可以扩展,即再增加两条边与两个顶点(见图 7-16)。

现在不管用 7.1 节和本节所讲的办法,都无法再扩大 T_1 了,因此,可以断定以 V_{10}为一端的可增广轨一定不存在。因此,我们应该再取一个未盖点,例如 V_{11},然后来扩大以 V_{11}为根的交错树 T(见图 7-17(a))。

由于 T 中存在一条由 V_{14}为一端的可增广轨,因此在这条可增广轨上用粗细对换的办法把图 7-14 中的匹配 M_2 改成图 7-17(b)中的匹配 M_3。

对于匹配 M_3 来说,只有一个未盖点 V_{10},因此 M_3 一定是一个最大匹配了。

扩展交错树 T_1

图 7-16

以V_{11}为根的交错树

(a)

粗边为 M_3

(b)

图 7-17

二、程序

下面,我们给出求任意图的最大匹配的程序。

```
  program ren_yi_tu_de_zui_da_pi_pei;
{ 任意图的最大匹配 }
    uses
      crt;
    const
      maxn              = 50;
    type
      graphtype         = array[1..maxn,1..maxn] of integer; { 图类型 }
      listtype          = array[1..maxn,1..2] of integer;
      ltype             = array[1..maxn] of integer;
    var
      g                 : graphtype;                  { 图的邻接矩阵 }
      l                 : listtype;     { l[i,2]—与 i 顶点相连的盖点序号 }
                                        { l[i,1]—与 i 顶点相连的未盖点序号 }
      p
: ltype;      {         ┌=0 顶点 i 为未盖点                       }
              {p[i]=┤                                            }
              {         └≠0 顶点 i 为盖点,<i,p[i]>为匹配边;     }
      n                 : integer;                              { 顶点数 }
      f                 : text;                               { 文件变量 }
    procedure init;
      var i,j : integer;
          str : string;
        begin
          clrscr;
          write('file name = '); { 输入文件名,并与文件变量连接 }
          readln(str);
          assign(f,str);
          reset(f);
```

```
      readln(f,n); { 读入顶点数 }
      fillchar(g,sizeof(g),0);     { 图矩阵初始化 }
      while not eof(f) do          { 读入邻接矩阵 }
        begin
          readln(f,i,j);
          g[i,j]:=1;
          g[j,i]:=1;
        end;
      close(f); { 关闭文件 }
    end;
procedure findway(i,j:integer);
{ 将可扩展路上的匹配边与非匹配边对换 }
  var q,k : integer;
    begin
      q:=i; k:=j; p[j]:=i;
      while l[q,2]<>maxint do
        begin
          p[q]:=k; k:=l[q,2];
          p[k]:=l[k,1];
          q:=l[k,1];
        end;
      p[q]:=k;
    end;
function kuo_tree:boolean;  { 若扩展路存在,返回 true, 否则返回 false }
  var i,j,q,k,v : integer;
      s : set of 1..maxn;
      more : boolean;
    begin
      kuo_tree:=true;
      repeat
        more:=false;        { 设扩展路搜索成功标志 }
        for i:=1 to n do
          if l[i,2]>0 then { 若已确定与 vi 相连的一条匹配边 }
            for j:=1 to n do
              if (g[i,j]>0) and (p[i]<>j) then
                 { vi 与 vj 有未检查的边且非匹配边 }
                if (l[j,1]=0) and (l[j,2]=0)
                 { 若未确定 Vj 的匹配边和非匹配边 }
                  then if p[j]=0 { 若 vj 是未盖点 }
                       then begin
                              findway(i,j);
                     { 找着可扩展路,将该路上的匹配边与非匹配边对换 }
                              exit;
                            end
                       else begin { vj 是盖点 }
                              g[i,j]:=-g[i,j]; g[j,i]:=-g[j,i];
                            { 设非匹配边<vi,vj>检查标志 }
                              l[j,1]:=i; l[p[j],2]:=j;
                            { 将匹配边<j,p[j]>边入 T 中,设该边检查标志 }
```

```
                            g[j,p[j]]:=-g[j,p[j]];
                            g[p[j],j]:=-g[p[j],j];
                            more:=true;{ 继续搜索扩展路 }
                          end
              else if (l[j,1]=0) and (l[j,2]>0) then
            { 若与 vj 相连的匹配边确定而非匹配边未确定,即找到一 }
            { 条不属于 T 且连接 T 上两个外点的边<vi,vj> }
                    begin
                      more:=true;{ 继续搜索扩展路 }
                      g[i,j]:=-g[i,j]; g[j,i]:=-g[j,i];
                      { vi 与 vj 是检查标志 }
                      s:=[i];   k:=i;   v:=2;
                      while l[k,v]<>maxint do { 搜索收缩顶点 k }
                        begin k:=l[k,v]; s:=s+[k]; v:=3-v; end;
                      k:=j; v:=2;
                      while not (k in s) do
                        begin k:=l[k,v]; v:=3-v; end;
                      if l[i,1]=0 then
                        begin
                          l[i,1]:=j; q:=i; v:=2;
                          { 连接边<vi,vj>,建立途经收缩花内各 }
                          { 顶点间的一条交错路 }
                          while q<>k do
                            begin
                              l[l[q,v],v]:=q; q:=l[q,v]; v:=3-v;
                            end;
                        end;
                      l[j,1]:=i; q:=j; v:=2;
                      while q<>k do
                        begin
                          l[l[q,v],v]:=q; q:=l[q,v]; v:=3-v;
                        end;
                    end;
    until not more;
    kuo_tree:=false;  { 扩展路不存在标志 }
  end;

procedure main;
  var i,j,k,tot : integer;
      b : boolean;
  begin
    fillchar(p,sizeof(p),0); tot:=0;
    repeat i:=0;
          repeat inc(i);   { 考察下一个顶点 }
                if p[i]=0 then { 顶点 vi 是未盖点 }
                  begin
                    fillchar(l,sizeof(l),0);              { T 树初始化 }
                    l[i,2]:=maxint;          { 搜索以 i 顶点为一端的扩展路 }
                    b:=kuo_tree;
                    for j:=1 to n do            { 恢复初始图矩阵 }
```

```
                    for k:=1 to n do
                      g[j,k]:=abs(g[j,k]);
                  end
                else b:=false; { vi 为一端的扩展路不存在 }
          until (i>n) or (b);
          { 直至求出最大匹配或以 vi 为一端的扩展路存在为止 }
          if i<=n then inc(tot);
          { 若找到以 Vi 为一端的扩展路,则匹配边数加 1 }
      until (i>n) or (tot=n div 2);{ 直至求出最大匹配或完全匹配为止 }
      for i:=1 to n do { 打印匹配方案 }
        if p[i]>0 then
          begin writeln(i,'——',p[i]); p[p[i]]:=0; end;
      writeln('number of pi pei : ',tot);
    end;

begin
  init; { 输入任意图 }
  main; { 计算和输出最大匹配 }
end.
```

7.6 求最小边的覆盖

一、问题及其算法

图 7-18 是一个公路网,图 G 中的顶点看成是一些村庄,每条边看成是一段公路。如果在一段公路上放上一辆消防车,就可以把这段公路两端的两个村庄控制起来了,现在问至少用几辆消防车才能把所有村庄全控制住?这些消防车应该放在哪些公路上?

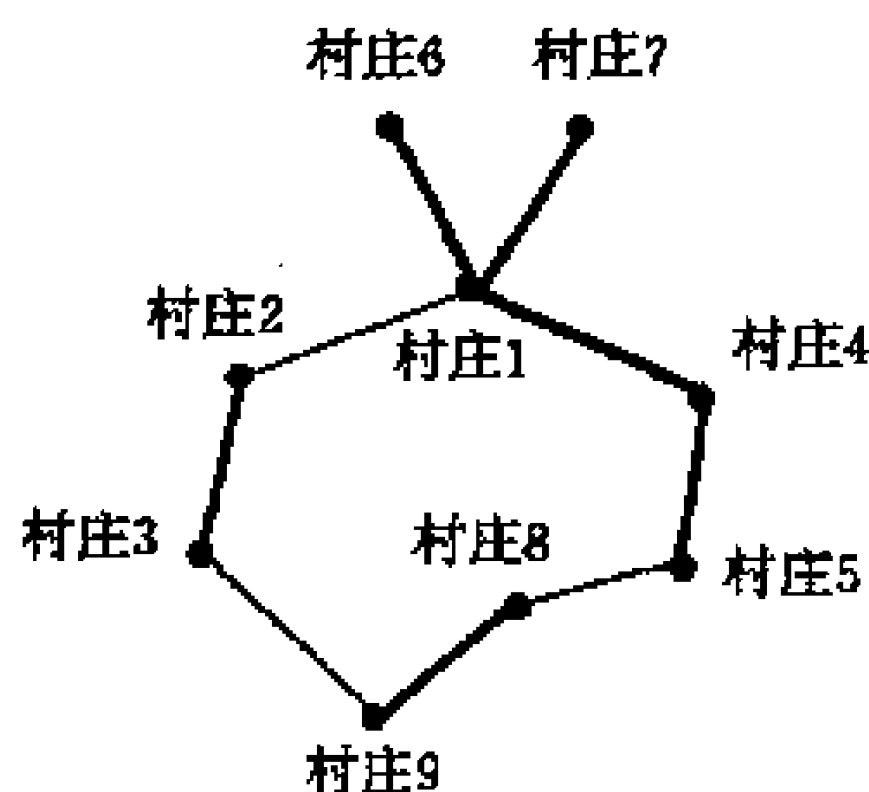


图 7-18

上述问题的数学模型是:

求无向图 $G=[V,E]$的一个边集合,这个边集合恰好把图的所有顶点都盖住且含边数最少。这就是所谓的最小边覆盖。

例如图 7-18 中,粗边组成的集合就是一个最小边覆盖。6 辆消防车放在这些边对应的公路上,就能把所有村庄全控制住。

由于一条边只能盖住两个顶点,因此对于一条有 N 个顶点的图 G 来说,盖住所有顶点的边数不会少于 $N/2$ 条,即最小边覆盖中的边数至少为 $N/2$ 条。是不是任何图都有最小边覆盖呢?不一定!因为只要图中有不和任何边关联的孤立顶点,这个图就不可能有边覆盖了。不过很容易看出,只要图 G 没有孤立顶点,G 就一定有边覆盖(因为这时,图中所有边的集合就是一个边覆盖),就可以从图 G 的所有边覆盖中,将含边数最少的边覆盖找出来。

求最小边覆盖的算法的基本思想,其实是很简单的。我们希望选取尽量少的边把一个图的所有顶点都盖住,第 1 条边不管怎样选,一定恰好盖住两个顶点,第 2 条边如果选

得与第 1 条边没有公共端点，就又可以多盖住两个顶点，而如果与第 1 条边有公共端点，实际上只多盖住了一个顶点。一般说来，如果选定了 S 条边，这些边两两都没有公共端点，能盖住的顶点就最多，即盖住了 $2S$ 个，从而选取的边也就可能最少。因此我们应该尽量多选一些两两没有公共端点的边来覆盖，很自然的应该先选取一个最大匹配 M，因为 M 中含两两没有公共端点的边数最多。剩下那些未盖点，想要用一条边盖住两个顶点就不可能了，不得不对一个顶点用一条边来盖。由此得出算法步骤：

求最大匹配 M，得两两没有公共端点的 S 条边

↓

对每一个未盖点任取一条边与之相连，得 $N-2*S$ 条边

↓

M+盖住未盖点的 $N-2*S$ 条边为最小边覆盖

二、求最小边覆盖的程序

```
program zui_xiao_bian_fu_gai;
{ 最小边覆盖 }
  uses
    crt;
  const
    maxn          = 50;
  type
    graphtype     = array[1..maxn,1..maxn] of integer;
    listtype      = array[1..maxn,1..2] of integer;
    ltype         = array[1..maxn] of integer;
  var
    g             : graphtype; { 邻接矩阵 }
    l             : listtype;  { l[i,1]与 i 顶点相连的未盖点的序号        }
                               { l[i,2]与 i 顶点相连的盖点序号            }
                  : ltype;     {        ┌=0  顶点 i 为未盖点;}
    p                          {p[i]=┤                                     }
                               {        └≠0 顶点 i 为盖点,<i,p[i]>为匹配边;  }
    n             : integer;   { 顶点数 }
    f             : text;      { 文件变量 }
  procedure init;
    var i,j : integer;
        str : string;
      begin
        clrscr;
        write('file name = '); { 读入文件名,并与文件变量连接 }
        readln(str);
        assign(f,str);
        reset(f);
        readln(f,n);            { 读入顶点数 }
        fillchar(g,sizeof(g),0);
        while not eof(f) do    { 读入邻接矩阵 }
```

```
        begin
          readln(f,i,j);
          g[i,j]:=1;
          g[j,i]:=1;
        end;
      close(f);
    end;
procedure findway(i,j:integer);{将可扩展路上的匹配边与非匹配边对换 }
  var q,k : integer;
    begin
      q:=i; k:=j; p[j]:=i;
      while l[q,2]<>maxint do
        begin
          p[q]:=k; k:=l[q,2];
          p[k]:=l[k,1];
          q:=l[k,1];
        end;
      p[q]:=k;
    end;
function kuo_tree:boolean; { 若扩展路存在,返回 true, 否则返回 false }
  var i,j,q,k,v : integer;
      s : set of 1..maxn;
      more : boolean;
    begin
      kuo_tree:=true;
      repeat
        more:=false; { 设扩展路搜索成功标志 }
        for i:=1 to n do
          if l[i,2]>0 then { 若已确定与 vi 相连的一条匹配边 }
            for j:=1 to n do
              if (g[i,j]>0) and (p[i]<>j) then
                { vi 与 vj 有未检查的边且非匹配边 }
                if (l[j,1]=0) and (l[j,2]=0)
                { 若未确定 vj 的匹配边和非匹配边 }
                 then if p[j]=0 { 若 vj 是未盖点 }
                      then begin
                             findway(i,j);
                  { 找着可扩展路,将该路上的匹配边与非匹配边对换 }
                             exit;
                           end
                      else begin  { vj 是盖点 }
                             g[i,j]:=-g[i,j]; g[j,i]:=-g[j,i];
                { 设非匹配边<vi,vj>检查标志,并将该边加入 T 中 }
                             l[j,1]:=i; l[p[j],2]:=j;
                             g[j,p[j]]:=-g[j,p[j]];
                             g[p[j],j]:=-g[p[j],j];
                             more:=true; { 继续搜索扩展路 }
                     { 将匹配边<j,p[j]>边入 T 中,设该边检查标志 }
                           end
```

```
                else if (l[j,1]=0) and (l[j,2]>0) then
              { 若与 vi 相连的且匹配边确定而非匹配边未确定,即找 }
              { 出一条不属于 T 且连接 T 上的两个外点的边<vi,vj> }
                    begin
                      more:=true; { 继续搜索扩展路}
                      g[i,j]:=-g[i,j]; g[j,i]:=-g[j,i];
                      { <vi,vj>置检查标志 }
                      s:=[i];  k:=i;  v:=2; { 搜索收缩点 vk}
                      while l[k,v]<>maxint do
                        begin k:=l[k,v]; s:=s+[k]; v:=3-v; end;
                      k:=j; v:=2;
                      while not (k in s) do
                        begin k:=l[k,v]; v:=3-v; end;
                      if l[i,1]=0 then
                        begin
                          l[i,1]:=j; q:=i; v:=2;
                          { 连接边<vi,vj>,建立途经收缩花内各 }
                          { 顶点间的一条交错路 }
                          while q<>k do
                            begin
                              l[l[q,v],v]:=q; q:=l[q,v]; v:=3-v;
                            end;
                        end;
                      l[j,1]:=i; q:=j; v:=2;
                      while q<>k do
                        begin
                          l[l[q,v],v]:=q; q:=l[q,v]; v:=3-v;
                        end;
                    end;
      until not more;
      kuo_tree:=false; { 扩展路不存在标志 }
    end;
            { 直至求出最大匹配或以 vi 为一端的扩展路存在为止 }
            { 若找到以 vi 为一端的扩展路,则匹配边数加 1 }

procedure main;
  var i,j,k,tot : integer;
      b : boolean;
    begin
      fillchar(p,sizeof(p),0); tot:=0;
      repeat i:=0;
            repeat inc(i); { 考虑下一个顶点 }
                   if p[i]=0 then  { vi 是未盖点 }
                     begin
                       fillchar(l,sizeof(l),0);      { T 树初始化 }
                       l[i,2]:=maxint;
                       b:=kuo_tree;              { 搜索以 i 顶点为端的扩展路 }
                       for j:=1 to n do             { 恢复初始图矩阵 }
                         for k:=1 to n do
                           g[j,k]:=abs(g[j,k]);
```

```
                end
              else b:=false; { 以 vi 为一根的扩展路不存在 }
          until (i>n) or (b);
          { 直至求出最大匹配或以 vi 为一端的扩展路存在为止 }
          if i<=n then inc(tot);
          { 若找到以 vi 为一端的扩展路,则匹配数+1 }
    until (i>n) or (tot=n div 2);
    for i:=1 to n do
      if p[i]>0 { i 是盖点,则打印匹配边<i,p[i]> }
        then begin
               if i<p[i] then writeln(i,'——',p[i]);
             end
        else begin { i 是未盖点,任取一条与之相连的边<i,j>作覆盖边 }
               j:=0;
               repeat inc(j);
               until (j>n) or (g[i,j]<>0);
               if j>n
                 then writeln(i,' can not fu gai')
                 else begin writeln(i,'——',j); inc(tot); end;
             end;
    writeln('number of edges : ',tot);
  end;

begin
  init; { 输入任意图 }
  main; { 计算和输出最小边覆盖 }
end.
```

第八章　着色问题

8.1　求顶色数

一、考试安排问题

设学校共有 n 门功课需要进行期末考试，因为不少学生不止选修一门课，所以不能把一个同学选修的两门课安排在同一场次进行考试。问学期的期末考试最少需多少场次才能完成？

我们以每门功课为一个顶点，当且仅当两门功课被同一个学生选修时，在相应两个顶点之间连一条边，得到一个图 G（见图 8-1(a)）。

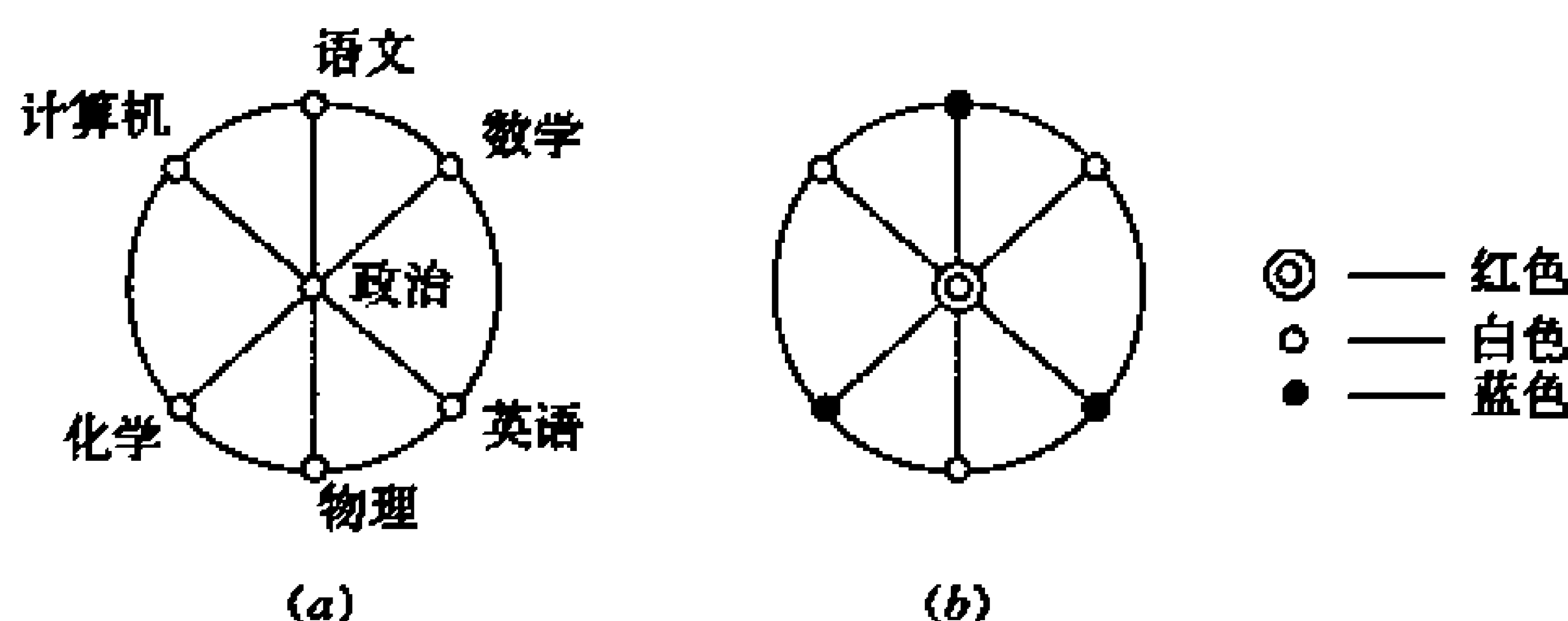


图　8-1

我们将 n 门功课划分成 K 个子集 $U_1, U_2, \cdots, U_k$，两两子集中的功课都不相同。每个子集 $U_i(1 \leqslant i \leqslant k)$ 中的顶点两两不相邻，即子集内的任两门功课都不能被一个学生选修。能按这种要求划分的子集数 K 必须最少，即不能将 G 的所有顶点再细划成 $k-1$ 个子集。然后我们对每一个子集内的顶点上一种颜色，同色顶点对应的课程安排在同一场次考试。于是颜色数 k 即为学期考试所需的最少场次数。

例如图 8-1(b)就是考试安排的一种方案。令◎顶点为第 1 场次考试科目，○顶点为第 2 场次考试科目，●顶点为第 3 场次的考试科目。显然

第 1 场：政治；

第 2 场：数学、物理、计算机；

第 3 场：语文、英语、化学。

考试安排问题，就是求顶色数的一个实际应用。

在讲算法之前，先讲一讲什么叫顶色数？

将图 G 的所有顶点涂上颜色，如果至少要用 K 种颜色方能使任意两个相邻的顶点颜色不同，我们就说 G 的顶色数为 K，记成 $X(G)=K$。

如果我们将同色的顶点列入一个顶点子集，那么求 $X(G)$，实际上是求满足下列条件

的最少子集数 K：

1. 两两子集中的顶点不同；

2. 每个子集中的两两顶点不相邻。

显然，平凡图的 $X(G)=1$，反过来 $X(G)=1$ 的图是平凡图；二分图的 $X(G)=2$（集合 X 与 Y 各用一种颜色），反过来 $X(G)=2$ 的图是二分图。对任意图 G 来说，$X(G)\leqslant\triangle+1$，即顶色数≤顶点度数最大值＋1。

我们由"每个同色顶点集合中的两两顶点不相邻"可以看出，同色顶点集实际上是一个独立集。当我们用第 1 种颜色上色时，为了尽可能扩大颜色 1 顶点的个数，逼近所用颜色数最少的目的，对 G 图的一个极大独立集涂颜色 1。用第 2 种颜色上色时，亦本着尽可能多些顶点上色的积极态度，选择另一个极大独立集涂色……，当所有顶点涂色完毕，所用的颜色数即为所选的极大独立集的个数。

当然上述颜色数未必就是 $X(G)$，而且其和能够含所有顶点的极大独立集个数未必唯一。于是我们必须从一切若干极大独立集的和含所有顶点的方案中，挑选所用极大独立集个数最小者，其个数即为所用的颜色数 $X(G)$。

由此可以得出算法步骤：

求 G 图的所有极大独立集；

求出一切若干极大独立集的和含所有顶点的方案；

从中挑选所用极大独立集个数最小者，即为 $X(G)$。

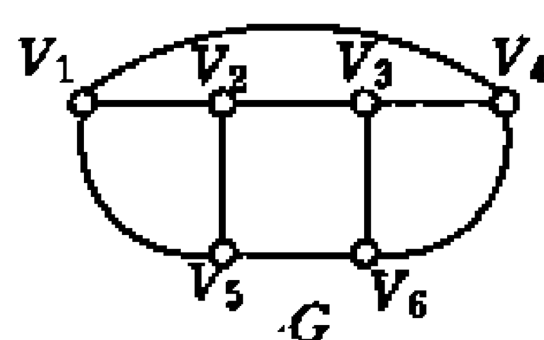


图 8-2

例如：求图 8-2 的 $X(G)$和着色方案。

图 G 的一切极大独立集如下：

$I_1=\{V_1,V_3\}$；　$I_2=\{V_2,V_4\}$；　$I_3=\{V_2,V_5\}$；

$I_4=\{V_5,V_1\}$；　$I_5=\{V_3,V_6\}$；　$I_6=\{V_4,V_6\}$。

显然我们可以选用 6 种颜色给每个顶点涂色，或者选用 5 种颜色分别给 5 个极大独立子集涂色，也可以选用 4 种颜色，例如为 I_1 中的 V_1,V_3 涂颜色 1，I_2 中的 V_2,V_4 涂颜色 2，I_3 中的 V_5 涂颜色 3，I_5 中的 V_6 涂颜色 1。

但上述方案的颜色数都不是 $X(G)$，正确的答案应该是 $X(G)=3$。有两种方案：一种方案是，给 I_1 中的 V_1 和 V_3 涂颜色 1，I_3 中的 V_2 和 V_5 涂颜色 2，I_6 中的 V_4 和 V_6 涂颜色 3；另一种方案留给读者自行分析。

由上可见，求色数其实需要求极大独立集以及一切若干极大独立集的和含所有顶点的方案。这对于大图，因为计算量过大而成为实际上难以奏效的算法，所以不是一个好算法。如果不强求答案精确的话，一般可以采用贪心法等近似算法来解。

二、程序

```
program se_su;
  uses
    crt;
  const
    maxn            = 50;
    maxs            = 100;
  type
    graphtype       = array[1..maxn,1..maxn] of integer;
    settype         = set of 1..maxn;
    listtype        = array[1..maxs] of settype;
    ltype           = array[1..maxn] of integer;
  var
    g               : graphtype; { 邻接矩阵 }
    l,l_a           : listtype;  { 极大独立集,辅助变量 }
    n,len,len_a,best : integer;  { 顶点数,极大独立集个数,辅助变量 }
                                 { 最少颜色数 }
    best_way        : settype;   { 被选中的极大独立集的序号集合 }
    cl              : ltype;     { cl[i]=1 第i个极大独立集被选中 }
    f               : text;      { 文件变量 }
  procedure init;
    var i,j : integer;
        str : string;
      begin
        clrscr;
        write('filename='); { 读文件名,并连接文件变量 }
        readln(str);        { 读准备,读顶点数 }
        assign(f,str);
        reset(f);
        readln(f,n);
        fillchar(g,sizeof(g),0);
        while not eof(f) do { 读邻接矩阵 }
          begin
            readln(f,i,j);
            g[i,j]:=1; g[j,i]:=1;
          end;
        close(f);
        best:=maxint;
      end;
  procedure add(s:settype);
  { 搜索l[1..len]。若s是l[i]是子集,则删去l[i];若l[i]是s的子集,}
  { 则退出;若l的所有集合与s非互为子集关系,则s进l表尾           }
    var i,j : integer;
      begin
        i:=1;
```

```
      while i<=len do
        if l[i]*s=l[i]
          then exit
          else if l[i]*s=s
               then begin
                      for j:=i to len-1 do
                        l[j]:=l[j+1];
                      dec(len);
                    end
               else inc(i);
      if len>maxs then
        begin
          writeln('overflow');
          halt;
        end;
      inc(len); l[len]:=s;
    end;
procedure du_li;      { 求图的极大独立集 l[1..len] }
  var i,j,k,t,v : integer;
      s         : settype;
    begin
      len:=1; l[1]:=[];
      for i:=1 to n do
        begin
          l_a:=l; len_a:=len; len:=0; s:=[];
          for j:=1 to n do
            if g[i,j]>0 then s:=s+[j];
          for k:=1 to len_a do
            begin
              add(l_a[k]+[i]);
              if s<>[] then add(l_a[k]+s);
            end;
        end;
      for i:=1 to len do l[i]:=[1..n]-l[i];
    end;
procedure findway(lev:integer);
  var i,j : integer;
    begin
      j:=0;
      for i:=1 to lev-1 do { 累计选中的独立子集,即颜色数 }
        if cl[i]=1 then inc(j);
      if j<best then { 若颜色数当前最少 }
        begin
          best:=j; best_way:=[];  { 则记下颜色数 }
          for i:=1 to lev-1 do     { 和被选中的极大独立子集的序号 }
            if cl[i]=1 then best_way:=best_way+[i];
        end;
    end;
```

```
procedure colour(lev:integer;done:settype);
  var x:settype;
      i:integer;
    begin
      if done=[1..n] { 若全部顶点正常上色,判别是否色数最少 }
        then findway(lev)
        else begin
               if not (l[lev]<=done)
               { 若当前上色的顶点是第 lev 个极大独立集的子集 }
                 then begin
                        cl[lev]:=1;
                        { 置第 lev 极大独立集的顶点上一种颜色的标志 }
                        colour(lev+1,done+l[lev]); { 继续递归搜索 }
                      end
                 else begin
                        x:=[]; { 求第 lev+1..len 的极大独立集的顶点和 }
                        for i:=lev+1 to len do x:=x+l[i];
                        if x+done=[1..n] then
               { 若已涂色的顶点与剩下顶点的和为全部顶点 }
                          begin
                            cl[lev]:=0;
                            { 说明第 lev 个极大独立子集不需涂色,置 }
                            { 该极大独立子集不涂色标志 }
                            colour(lev+1,done); { 继续递归搜索 }
                          end;
                      end;
             end;
    end;

procedure show;
  var i,j : integer;
    begin
      for i:=1 to len do { 若第 i 个极大独立子集被选中 }
        if i in best_way then
          begin
            for j:=1 to n do
            { 打印被涂一种颜色的第 i 个极大独立子集的顶点序号 }
              if j in l[i] then write(j:3);
            writeln;
          end;
      writeln('total number of colours : ',best); { 打印最少颜色数 }
    end;

begin
  init;          { 输入图 }
  du_li;          { 求极大独立集 }
  colour(1,[]); { 求顶点上色的最佳方案和顶色数 }
  show;           { 输出结果 }
end.
```

8.2 求边色数

8.2.1 边色数

一、边色数的概念及其算法

边色数是与顶色数相对应的概念。将图 G 的所有边涂上颜色，使得相邻的边互不相同，这时同一种颜色的边组成一个两两边互不相邻的边子集。若能将图 G 的边划分为这样的 K 个子集，则称 G 能正常 K 边着色。若至少要用 K 种颜色（即可以正常 K 边着色而不能 $K-1$ 边着色）时，则称 K 为 G 的边色数，记成 $X'(G)=K$。与顶点 U 关联的边中有 i 色边时，称 i 色在顶点 U 出现。在顶点 U 处出现的颜色数目记成 $C(U)$。

例如任何图皆可正常 $|E|$ 边着色。G 可以正常 K 边着色，当 $K<L\leqslant|E|$ 时，亦可正常 L 边着色。显然，$X'(G)\geqslant\triangle$。正常边着色时 $C(U)=d(U)$，其中 U 是图 G 中任取的一个顶点。$d(U)$ 为 U 的度数同色边的子集 $E_i(1\leqslant i\leqslant K)$ 是 G 中一个匹配。

数学家们曾对边色数的计算作了大量研究，引出不少有趣的结论：

1. G 是连通图，非奇圈（即所有圈含偶数个顶点），则存在 $X'(G)=2$，所使用的两种颜色在每个次数至少为 2 的顶点处出现；

2. X'（二分图）$=\triangle$；

3. G 是简单图（无环无重边），要么 $X'(G)=\triangle$，要么 $X'(G)=\triangle+1$。

有了求顶色数的基础，不难得出边色数的算法。首先通过边顶对换的方法将图 G 转换为图 G'：

G 图中的每条边 e 转换为图 G' 中的一个顶点 V'。若图 G 中两条边相邻，则 G' 中对应的两个顶点之间连一条边。然后对 G' 图求顶色数。在求 $X(G')$ 的过程中，应充分利用结论 3，使得选用的极大独立集的个数在$\triangle$和$\triangle+1$之间。最后将 $X(G')$ 个同色顶的集合转换为 G 图中同色边的集合。显然，$X(G')=X'(G)$。

例如：求图 8-3(a)的边色数。

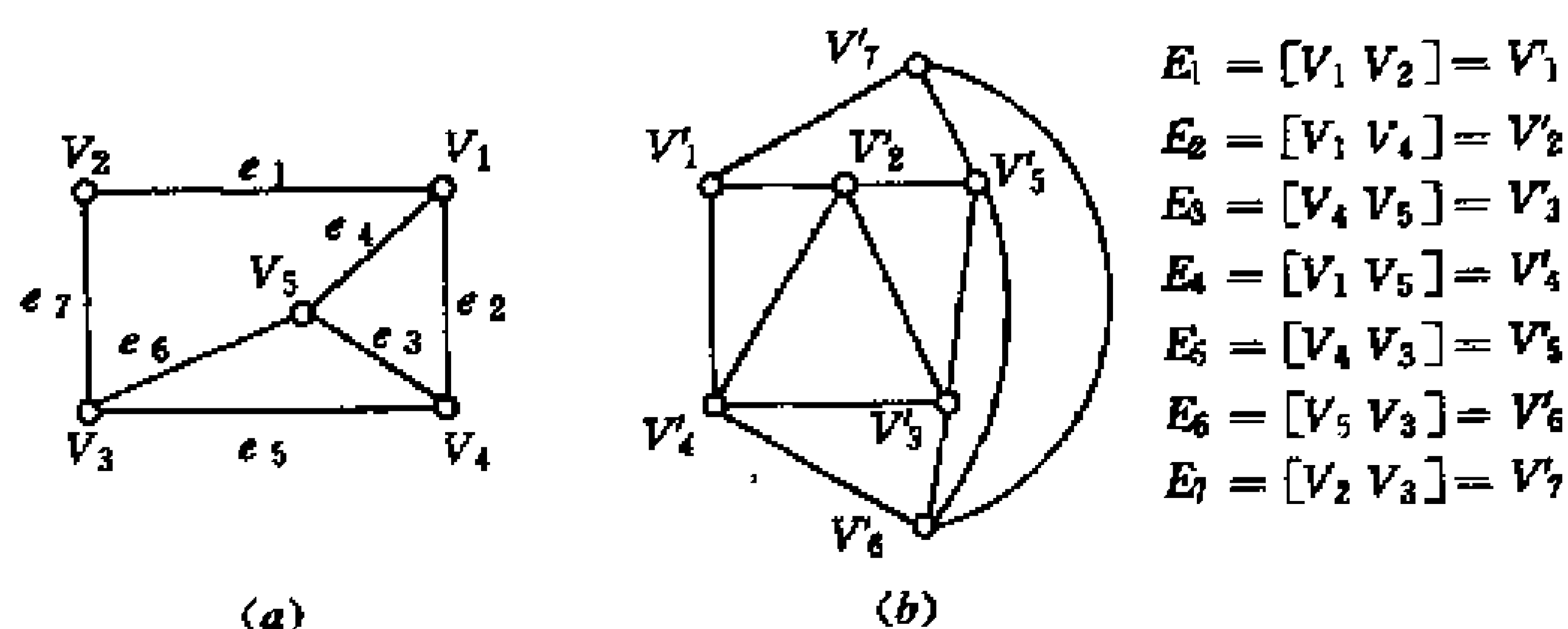


图 8-3

将图 G 按顶、边对换的方法转换为 G'（见图 8. 3(b)）。用最少颜色数给 G' 所有顶点上色的一种方案是（$\{V_2',V_6'\},\{V_4',V_5'\},\{V_1'\},\{V_3',V_7'\}$）。显然 $X(G')=4$。转换成对图

G 边上色（$\{E_2,E_6\},\{E_4,E_5\},\{E_1\},\{E_3,E_7\}$）。$X'(G)=4$。

二、求边色数的程序

```
{$M 65520,0,655360}
program dyeing_lines;
  const
    maxn                 =  200;
    { 最大独立集个数 }
  type
    settype              =  set of 1..maxn;
    linetype             =  record          { x 与 y 之间的一条边 }
                              x,y:byte
                            end;
    ltype                =  array [1..maxn] of linetype;
  var
    f                    :  text;
    n,del,lam,s,bd       :  byte;
    { 顶点个数、最大度数、边数、独立集个数、色数 }
    l                    :  ltype;
    { l[i]表示第 i 条边成为新图 G'的第 i 个顶点 }
    sn,dye,bdye          :  array [1..maxn] of settype;
    { 极大独立集、目前着色方案、最佳着色方案 }
  procedure init;
    var
      i,j,u,t:integer;
      str:string;
    begin
      write('File name = '); { 读入文件名，并与文件变量连接 }
      readln(str);
      assign(f,str);
      reset(f);
      readln(f,n);             { 读入顶点数 }
      lam:=0;del:=0;
      for i:=1 to n do { 读入邻接矩阵，并计算最大度 Δ }
        begin
          u:=0;
          for j:=1 to n do
            begin
              read(f,t);inc(u,t);
              if (t<>0) and (i<j)
                then begin
                       inc(lam);
                       l[lam].x:=i;l[lam].y:=j
                     end
            end;
          if u>del then del:=u
```

```
    end;
  close(f);
  s:=0;bd:=del+1 { 独立集个数初始化为 0,色数初始化为 Δ+1 }
end;
function check(w:settype;i:byte):boolean;
 { 若新图中顶点 i 与 w 中所有顶点相互独立,则返回 true,否则返回 false }
 var
   x:byte;
 begin
   check:=false;
   for x:=1 to lam do
     if (x in w) and ((l[i].x=l[x].x) or (l[i].y=l[x].y)
                 or (l[i].x=l[x].y) or (l[i].y=l[x].x)) then exit;
   check:=true
 end;
procedure find(w,w2:settype;k:byte);
 { 求所有极大独立集 }
 var
   i,j:integer;
   x:settype;
 begin
   if w=[1..lam]
      then begin
             inc(s);sn[s]:=w2
           end
      else for i:=k+1 to lam do
             if not (i in w)
                then begin
                       x:=[];
                       for j:=1 to lam do
                         if not check([i],j) then x:=x+[j];
                       find(w+x+[i],w2+[i],i)
                     end
 end;
procedure print(bd:byte);
 { 打印着色方案 }
 var
   i,j:integer;
 begin
   for i:=1 to bd do
     begin
       writeln('COLOR : ',i:3);
       for j:=1 to lam do
         if j in bdye[i] then writeln(l[j].x:3,'——',l[j].y:3);
       writeln('* * * * * * * * * * * *')
     end;
   readln;halt
 end;
```

```
procedure dyeing(w:settype;k,p:byte);
  { 求最佳着色方案 }
  var
   i:byte;
   x:settype;
  begin
    if w=[1..lam]
       then begin
                bdye:=dye;
                if p-1=del
                   { 由于输入的图是简单图，所以边色数或是 Δ，或是 Δ+1 }
                   { 若已经找到色数为 Δ 的方案，则打印结果并结束，否则继续 }
                   then print(del)
                   else dec(bd)
            end
       else begin
                if p>bd then exit; { 正常边色数大于 Δ+1，则退出 }
                if not (sn[k]<=w)
{ 若已涂色的顶点是第 k 个极大独立集的子集，则第 k 个极大独立集去除 }
                   then begin
{ 除去已涂色的顶点后的剩余顶点上第 p 种颜色，并继续递归搜索 }
                          dye[p]:=sn[k]-w;
                          dyeing(w+dye[p],k+1,p+1)
                        end;
                x:=sn[k]-w;
{ 若已涂色的顶点与剩下的顶点和为全部顶点，说明第 k 个极大独立集不需涂色，}
{ 继续递归搜索第 k+1 个极大独立集上第 p 种颜色的可能性                    }
                for i:=k+1 to s do x:=x-sn[i];
                if x=[] then dyeing(w,k+1,p)
            end
  end;

 begin
   init;             { 输入图 G，按边、顶对换的方法将 G 转换为图 G' }
   find([],[],0);    { 求 G'的所有极大独立集 }
   dyeing([],1,1);   { 求 G'的最佳顶点着色方案 }
   print(del+1)      { 打印对应于 G 的最佳边着色方案 }
 end.
```

8.2.2 边色数的一个实际应用

边色数的实际应用很广，下面举一个实例。

一、时间表问题

学校有 m 位教师 $X_1,X_2,\cdots,X_m$ 和 n 个班级 $Y_1,Y_2,\cdots,Y_n$，X_i 老师为 Y_j 班每天上课 P_{ij}学时，试安排一个授课表，使学校上课的时间最少。

令 $X=\{X_1,X_2,\cdots,X_m\}$，$Y=\{Y_1,Y_2,\cdots,Y_n\}$，顶点 X_i 与 Y_j 之间有 P_{ij}条边相连，形成一个二分图。每一学时，教师最多为一个班上课，每个班也至多接受一个老师的授课。所

以我们的问题的解就是求二分图的边色数。一个同色边集为同一课时里授课的教师和班级，反映教师与班级之间授课关系的边两两不相邻；不同色的边集之间表示不同课时的授课安排。

因 X'(二分图)$=\Delta$，若没有授课多于 P 节课的教师，也没有授课多于 P 节课的班级 $P=\max\{P_{ij}|1\leqslant i\leqslant m,1\leqslant j\leqslant n\}=\Delta$，则可以编出一个至多 P 节课的时间表。

例如：4 名教师、5 个班级，教学要求如下：

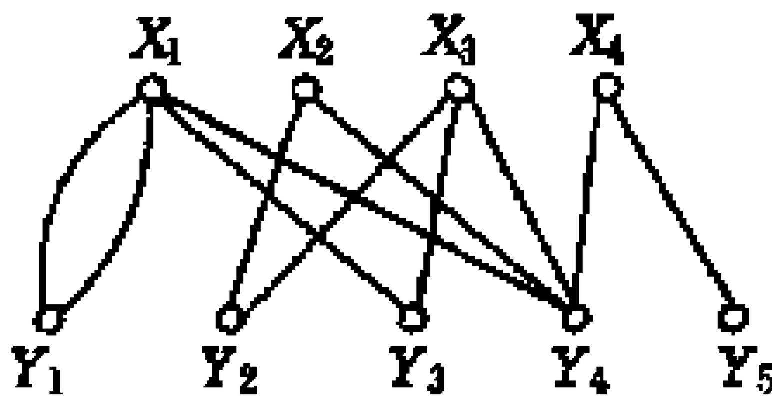


图 8-4

$$A=\begin{array}{c} \\ X_1 \\ X_2 \\ X_3 \\ X_4 \end{array}\begin{array}{c} \begin{array}{ccccc} Y_1 & Y_2 & Y_3 & Y_4 & Y_5 \end{array} \\ \begin{bmatrix} 2 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \end{array}$$

以 $X=\{X_1,X_2,X_3,X_4\}$，$Y=\{Y_1,Y_2,Y_3,Y_4,Y_5\}$为二分图的顶点构作二分图，当 A 矩阵中 i,j 号元素为 a_{ij}时，则在 X_i 与 Y_j 之间加 a_{ij}条边(见图 8-4)。

按上一节所讲的算法，得出边色数 $X'(G)=4$，其中一种课表方案如表 8-1 所示。

表 8-1

节 / 教师	1	2	3	4	5	6
X_1	Y_1	Y_1	Y_3	Y_4	—	—
X_2	Y_2	—	Y_4	—	—	—
X_3	Y_3	Y_4	—	Y_2	—	—
X_4	Y_4	Y_5	—	—	—	—

共用四间教室，因为第一节课 Y_1,Y_2,Y_3,Y_4 四个班都在上课。如果学校的设施有限，要减少若干教室，令学校一天最少安排几节课？换句图论的话说，题意要求解出一种边色数 $P>\Delta$ 的正常着色，(即增加每天的课时数)，使得同色边集中的边数减少(即减少同课时内授课的教师数)，以达到减少教室数的目的。显然，要开设 $L(L=\Sigma P_{ij}(1\leqslant i\leqslant m,1\leqslant j\leqslant n))$ 门功课，编成每天 $P(P\geqslant\Delta)$节课的功课表，每节课平均要开 L/P 门功课，至少需要$\{L/P\}$间教室。为了编制功课表，我们首先必须求出边色数 P：

$$P=\begin{cases}\Delta & [L/\text{教室数}]+1<\Delta \\ [L/\text{教室数}] & (L \bmod \text{教室数}=0)\ \text{AND}\ ([L/\text{教室数}]+1\geqslant\Delta) \\ [L/\text{教室数}]+1 & (L \bmod \text{教室数}<>0)\ \text{AND}\ ([L/\text{教室数}]+1\geqslant\Delta)\end{cases}$$

二、排时间表的程序

下面给出时间表问题的程序题解：

```
program time_table_arrangment;
  const
    maxn              =  255;
```

```
  { 最多课程数 }
type
  linetype          =  record
                         x,y:byte { 第 x 个教师上 y 班的一节课 }
                       end;
  ltype             =  array [1..maxn] of linetype;
                       { 边顶对换形成的新图 }
  settype           =  set of 1..maxn;
var
  m,n,p,lam,sn,bd,kk   :  integer;
  { 教师数、班级数、教室数、总课时、极大独立集个数、每天的最大课数、Δ 值 }
  f                    :  text;
  l                    :  ltype;
  s,men                :  array [1..maxn] of settype;
  { 极大独立集,最佳排课方案,(当前排课方案) }
procedure init;
  var
    i,j,t,x,u:byte;
    str:string;
  begin
    write('File name = '); { 输入文件名,并与文件变量连接 }
    readln(str);
    assign(f,str);
    reset(f);
    readln(f,m,n,p);        { 读入教师数、班级数和教室数 }
    lam:=0;kk:=0;             { 总课时和每天最大课数初始化 }
    for i:=1 to m do
      begin
        u:=0;
        for j:=1 to n do
          begin
            read(f,t); { 读入教学要求 }
            for x:=lam+1 to lam+t do
              begin
                l[x].x:=i;l[x].y:=j
              end;
            lam:=lam+t; { 计算总课时数 }
            inc(u,t)
          end;
        if u>kk then kk:=u { 计算每天最大课数 }
      end;
    close(f);
    sn:=0;
  end;
function check(w:settype;k:byte):boolean;
  { 检查 k 是否相对 w 中的所有元素独立 }
  var
    i:integer;
```

```
  begin
    check:=false;
    for i:=1 to lam do
      if (i in w) and ((l[k].x=l[i].x) or (l[k].y=l[i].y))
          then exit;
    check:=true
  end;
procedure find(d,k:byte;w:settype);
  { 求极大独立集 }
  var
    i:integer;
    b:boolean;
  begin
    b:=false;
    if d<p
       then for i:=k+1 to lam do
                 if not (i in w) and check(w,i)
                    then begin
                               b:=true;
                               find(d+1,i,w+[i])
                           end;
    if not b
       then begin
                 for i:=1 to sn do if w<=s[i] then exit;
                 inc(sn);
                 s[sn]:=w
               end
  end;
procedure print;
  { 打印排课方案 }
  var
    i,j:integer;
  begin
    writeln;
    for i:=1 to bd do
      begin
        writeln('*****',i,'*****');
        for j:=1 to lam do
          if j in men[i]
             then writeln('TEACHER: ',l[j].x:4,' CLASS: ',l[j].y:4);
      end;
    readln;halt
  end;
procedure paint(pp,k:integer;w:settype);
  { 对新图1求正常着色方案 }
  var
    i:integer;
    x:settype;
```

```
begin
  if w=[1..lam]
    then begin
           if pp-1<=bd then print{ 若目前方案是正常着色方案，则打印结果 }
         end
    else begin
           if pp>bd then exit;
           if not (s[k]<=w)
             then begin
                    men[pp]:=s[k]-w;
                    paint(pp+1,k+1,w+men[pp])
                  end;
           x:=s[k]-w;
           for i:=k+1 to sn do x:=x-s[i];
           if x=[] then paint(pp,k+1,w)
         end
end;
begin
  init;
  find(0,0,[]); { 求极大独立集 }
  if lam div p+1 < kk { 根据总课时和教室数求每天的最大课时数 bd }
    then bd:=kk
    else if lam mod p = 0
           then bd:=lam div p
           else bd:=lam div p + 1;
  paint(1,1,[]); { 求 bd 边正常着色 }
end.
```

三、另一种算法

下面，我们给出一个新的算法，这个算法是由两条定理引出的：

1. E_i、E_j 是对图 G 正常着色时的两个同色边集（显然 E_i，E_j 是图 G 的无公共边的匹配），且 $|E_i|>|E_j|$，则存在对 G 正常着色的另两个同色边集 E_i'，E_j'，使得

$$|E_i'| = |E_i| - 1; |E_j'| = |E_j| + 1$$
$$E_i' \cup E_j' = E_i \cup E_j$$

那么，如何求满足上述条件的 E_i'，E_j' 呢？我们给出一种方法：

在由 E_i 与 E_j 组成的子图里，寻找一条边在 E_i 与 E_j 中交错出现且始边和终边皆在 E_i 内的 P 中：

$$P=e_1,e_2,\cdots,e_{2n+1}$$

其中 $e_1e_3\cdots e_{2n+1}$ 为 E_i 中的边，用细实线表示；

$e_2e_4\cdots e_{2n}$ 为 E_j 中的边，用粗实线表示。

我们采用粗细实线易位的办法得出

$E_i'=(E_i-\{e_1e_3\cdots e_{2n+1}\})\cup\{e_2,e_4\cdots,e_{2n}\}$

$E_j'=(E_j-\{e_2e_4\cdots e_{2n}\})\cup\{e_1,e_3\cdots,e_{2n+1}\}$

显然E'_1与E'_4亦是对G正常着色的两个同色边集,满足定理1的要求。

例如,第1节课的授课安排E_1与第4节课的授课安排E_4有一条交错轨$(Y_1X_1)(X_1Y_4)(Y_4X_4)$(见表8. 1)。我们将这条轨粗细线易位,得出$E'_1=\{(X_1Y_4)(X_2Y_2)(X_3Y_3)\}$,$E'_4=\{(X_1Y_1)(X_3Y_2)(X_4Y_4)\}$(见表8-2)。

表 8-2

节 教师	1	2	3	4	5	6
X_1	Y_4	Y_1	Y_3	Y_1	—	—
X_2	Y_2	—	Y_4	—	—	—
X_3	Y_3	Y_4	—	Y_2	—	—
X_4	—	Y_5	—	Y_4	—	—

共用3间教室,因为每节课至多3个班级在上课。同样,我们可以看出,要排出在4节课里完成$L=11$门功课的课表,至少需要3间教室。那么,是否可以通过适量增加一天课时数的办法减少教室呢?我们引出定理2:

2. G是二分图,$\Delta\leqslant P$,则存在P个对G图正常着色的同色边集$E_1,E_2,\cdots,E_P$(显然$E_1,E_2,\cdots,E_P$是图G内P个无公共边的匹配),使得

$$E(G)=\bigcup_{i=1}^{P}E_i$$

且对于$1\leqslant i\leqslant P$

$[G$的边数$L/P]\leqslant|E_i|\leqslant\{G$的边数$L/P\}$

那么如何通过适量增加一天的课时数$(P>\Delta)$来减少教室数$\{L/P\}$呢?将$E(G)$划分为$E_1,E_2,\cdots,E_\Delta,E_{\Delta+1},\cdots,E_P$,其中$E_1,E_2,\cdots,E_\Delta$为

对G_4边着色的同色边集,$E_{\Delta+1}=\cdots=E_P=\varphi$,使得$E(G)=\bigcup_{i=1}^{P}E_i$

反复运用定理1和2于那些边数差大于1的每对同色边集,我们则得到P个对G正常着色的同色边集$E'_1,E_2,\cdots,E'_P$,满足$[L/P]\leqslant|E_i|\leqslant\{L/P\}(1\leqslant i\leqslant P)$的要求。

例如,欲占用两个教室,由于$\{11/6\}=2,[11/6]=1$,则可编排每天6节课的课表,每节课1~2个班在上课。

表8-3的6节课的课表,是根据定理1和2,把表8-2中的同色边集调整成6个两两不相交的同色边集的结果。

表 8-3

节 教师	1	2	3	4	5	6
X_1	Y_4	Y_3	Y_1	—	Y_1	—
X_2	Y_2	Y_4	—	—	—	—
X_3	—	—	Y_4	Y_3	Y_2	—
X_4	—	—	—	Y_4	—	Y_5

这个程序留给读者编写。

第九章 可行遍性问题

可行遍性问题是指下述两类问题：

1. 是否可以从图 G 的任何一个顶点出发，不重复地行遍所有的边。如果可以，则图 G 中必有一条路(回路)，经过各条边恰好一次。这条路(回路)称为欧拉路(回路)。1.3 节中的一笔画问题就是欧拉路(回路)问题。

2. 是否可以从图 G 的任何一个顶点出发，不重复地行遍所有的顶点。如果可以，则图 G 中必有一条轨(圈)，经过每个顶点恰好一次。这条轨(圈)称为哈密尔顿轨(圈)。

从表面看，哈密尔顿轨(圈)与欧拉路(回路)非常相似，但实质上两者的理论迥然不同。在 1.3 节中给出了一个很简单的方法来判断一个图是否可以一笔画成，可是对于哈密尔顿轨(圈)，迄今为止尚无有效算法，也不知道究竟存在不存在有效算法。在这种情况下，我们只能介绍几种近似算法。

9.1 中国邮路问题

一、问题及其解法

一个邮递员从邮局选好邮件去投递，然后回到邮局。当然他必须经过他所管辖的每条街至少一次。请为他选择一条路线，使其所行路程尽可能地短。

我们首先把这个邮递员的辖区变成一个图(如图 9-1)，路口变成顶点，每条街变成边，边上的权表示街的长度。上述邮路问题就变成了在连通加权图上求取含有一切边的权最小的回路。我们称这种回路为理想回路。显然在欧拉图中，每条欧拉回路都是理想回路。

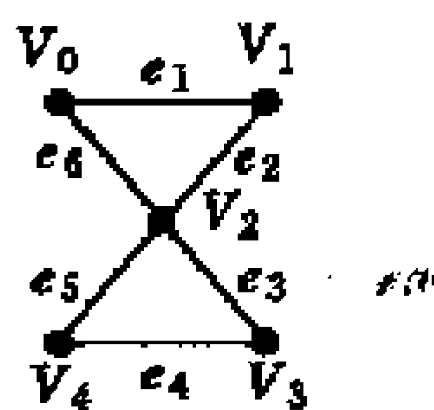
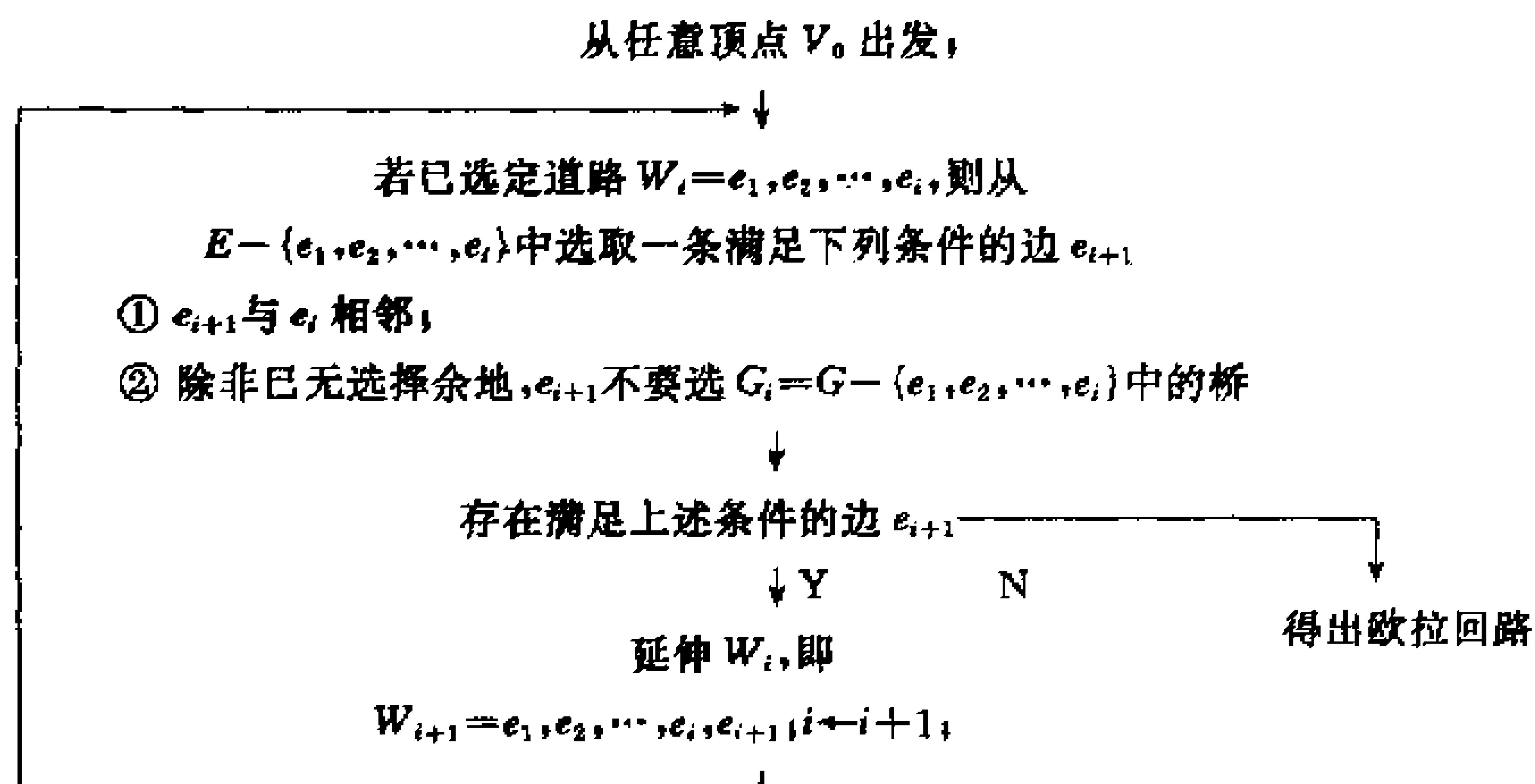


图 9-1

下面介绍一种在欧拉图中求欧拉回路的算法：

设 W_i——当前已选定的一条各边相异的路 $e_1, e_2, \cdots, e_i$。



按上述算法求图 9-1 中的欧拉回路：

从 V_0 出发，取 $W_1=e_1$。这时与 e_1 相邻的未采用过的边只有 e_2。尽管 e_2 是 $G-e_1$ 中的桥，但已无其它选择的余地，故只有取

$$W_2=e_1e_2$$

至此，W_3 不可选为 $e_1e_2e_6$，因为 e_6 是 $e-\{e_1e_2\}$ 的桥，而这时并非无其它选择的余地，例如可以不选 e_6 而选 e_3 和 e_5，所以可以取

$$W_3=e_1e_2e_3$$

继续执行上述算法，得

$$W_4=e_1e_2e_3e_4$$

$$W_5=e_1e_2e_3e_4e_5$$

$$W_6=e_1e_2e_3e_4e_5e_6$$

W_6 即为欧拉回路。

二、程序

下面给出求欧拉图中欧拉回路的程序。

```
program fleury;
  const
    maxn        =  30;
  type
    nodetype     =  record  { 可增广轨的类型：标号和检查标志 }
                      l,p:integer;
                    end;
    arctype      =  record  { 网结点类型：容量和流量 }
                      c,f:integer
                    end;
    gtype        =  array [1..maxn,0..maxn] of arctype;
    ltype        =  array [1..maxn] of nodetype;
```

```
var
  f            :  text; { 文件变量 }
  g            :  gtype;    { 图的邻接矩阵 }
  n,s,t        :  integer;  { 顶点数、源点、汇点 }
  lt           :  ltype;    { 可改进路 }
function read_graph:integer; { 初始化:读入邻接矩阵 g,并返回奇顶点的个数 }
  var
    str:string;
    i,j,t:integer;
  begin
    write('Graph file = '); { 读入文件名,并与文件变量连接 }
    readln(str);
    assign(f,str);
    reset(f);
    readln(f,n);              { 读入顶点数 }
    fillchar(g,sizeof(g),0);
    t:=0;
    for i:=1 to n do
      begin
        for j:=1 to n do
          begin
            read(f,g[i,j].c);inc(g[i,0].c,g[i,j].c)
            { 读入邻接矩阵,计算 i 顶点的度 }
          end;
        if odd(g[i,0].c) then inc(t) { 计算奇顶点的个数 }
      end;
    close(f);
    read_graph:=t
  end;
function check:integer; { 返回一个已标号而未检查的顶号序号 }
  var
    i:integer;
  begin
    i:=1;
    while (i<=n) and not ((lt[i].l<>0) and (lt[i].p=0)) do inc(i);
    if i>n then check:=0
           else check:=i
  end;
function ford(var a:integer):boolean;
{若存在 vs 至 vt 的可改进路返回 false 和改进量 a }
  var
    i,j,m,x:integer;
  begin
    ford:=true;
    fillchar(lt,sizeof(lt),0); { 改进路初始化 }
    lt[s].l:=s;        { 从 vs 开始搜索 }
    repeat
      i:=check;        { 找一个已标志而未检查的顶点序号 }
```

```
      if i=0   then exit;   { 该顶点若不存在,则无可改进路,返回 true }
      for j:=1 to n do      { 置所有与 i 相邻的弧的前向弧或后向弧标志 }
        if (lt[j].l=0) and ((g[i,j].c<>0) or (g[j,i].c<>0))
           then begin
                  if (g[i,j].f<g[i,j].c) then lt[j].l:=i;
                  if (g[j,i].f>0) then lt[j].l:=-i
                end;
      lt[i].p:=1     { 置 i 结点已检查标志 }
    until (lt[t].l<>0); { 直至 vs 至 vt 的一条改进路找出为止 }
    m:=t;a:=maxint;     { 从 vt 开始,倒向计算可改进量 }
    repeat
      j:=m;m:=abs(lt[j].l);
      if lt[j].l<0 then x:=g[j,m].f-0;
      if lt[j].l>0 then x:=g[m,j].c-g[m,j].f;
      if a>x then a:=x
    until m=s;
    ford:=false { 返回可改进路存在标志 false }
  end;

procedure fulkerson(a:integer);
{ 输入可改进量 a,从 vt 开始沿可改进路,逐弧修正流量 }
  var
    m,j:integer;
  begin
    m:=t;
    repeat
      j:=m;m:=abs(lt[j].l);
      if lt[j].l<0 then g[j,m].f:=g[j,m].f-a;
      if lt[j].l>0 then g[m,j].f:=g[m,j].f+a
    until m=s
  end;

function proceed(a,b:integer):boolean;
  var
    p,i,del:integer;
    success:boolean;
  begin
    for s:=1 to n do
      for t:=1 to n do
        if (s<>t) and (g[s,t].c=0)
        { 若 vs 与 vt 间无弧,则以 vs 为源、vt 为汇求最大流 }
        then begin
               for i:=1 to n do { 所有弧的流量初始化 }
                 for p:=1 to n do g[i,p].f:=0;
               repeat
                 success:=ford(del);
                 { 搜索可改进路和改进量 del }
                 if not success then fulkerson(del)
                 { 若可改进路存在,则按 del 修正该路上的流量 }
               until success; { 直至最大流求出 }
               p:=0;for i:=1 to n do p:=p+g[s,i].f;
```

```
                { 求最大流量 p }
                if (p=1) and (g[a,b].f=1)
                    { 若最大流量为 1 且 va 至 vb 的流量为 1 }
                    then begin
                           proceed:=true;exit
                           { 则 va 和 vb 间的弧为桥，返回 true }
                         end
          end;
      proceed:=false
    end;

  procedure Euler;
    var
      i,x,t:integer;
    begin
      x:=read_graph; { 读入图矩阵,并返回奇顶点的个数 }
      if x<>0 then exit; { 存在奇顶点,则返回 }
      x:=1;                  { 从 v1 开始搜索 }
      repeat
        t:=maxint;       { 设以 x 为尾的弧不存在 }
        for i:=1 to n do
          if (g[x,i].c=1) { 若<x,i>有弧 }
             then if proceed(x,i) { 若<x,i>是桥,暂设 i 为弧头,继续循环 }
                     then t:=i
                     else begin
                            t:=i;i:=n { 若<x,i>非桥,设 i 为弧头,跳出循环 }
                          end;
        if t<>maxint { 以 x 为尾的弧存在 }
           then begin
                  writeln(x,'———',t); { 打印弧<x,t> }
                  g[x,t].c:=0;g[t,x].c:=0;x:=t
                  { 在图中撤去<x,t>,继续搜索以 t 为尾的下一条弧 }
                end
      until t=maxint; { 直至得出 Euler 回路 }
    end;

  begin
    Euler              { 求 Euler 回路 }
  end.
```

上述中国邮路问题是一种特定的邮路问题，即图 G 本身必须是欧拉图。

若 G 不是欧拉图，任何理想回路通过某些边多于 1 次。例如 9-2(a)中 $X-U-Y-W-U-Z-W-Y-X-U-W-U-X-Z-Y-X$ 是理想回路，边 UX，XY，YW 和 WY 上通过了两次。把边 e 的两端间再连上一条权亦为 $W(e)$的新边，则称新边为原来那条边的倍边。UX，XY，YW 和 WY 都有倍边(见图 9-2(b))。这样的邮路问题更具有通用性，故称一般邮路问题。

它的数学模型是：

1. 用倍边法由 G 得到 G^* 图，使 G^* 图是欧拉图，且

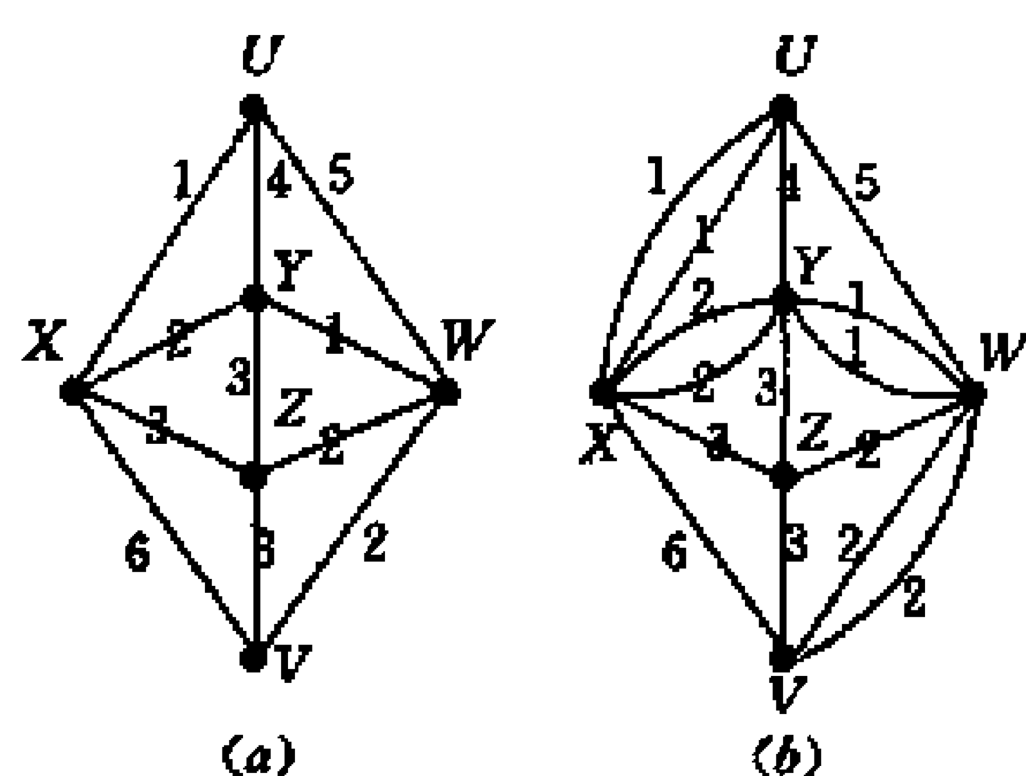


图 9-2

$$\sum W(\text{倍边}\ e) = \min$$

2. 在 G^* 中找到欧拉回路。
这个程序留给读者去编写。

9.2 货郎问题1

一、货郎问题与哈密尔顿回路问题

1859 年英国数学家哈密尔顿(Hamilton)提出了一种名为周游世界的游戏。他用一个正十二面体(图 9-3(*a*))的二十个顶点代表二十个大城市，要求沿着棱，从一个城市出发，经过每个城市恰好一次，然后回到出发点。

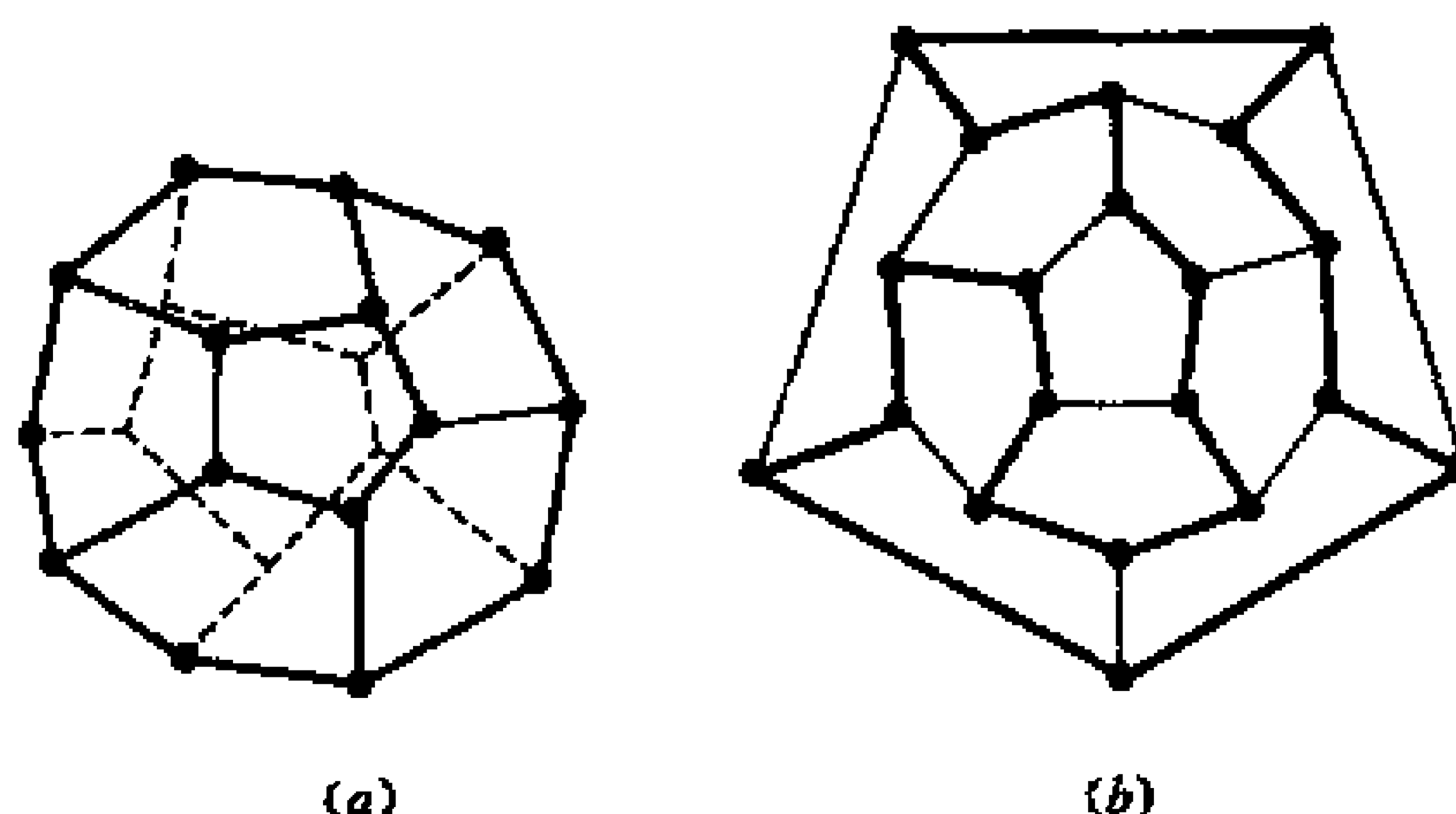


图 9-3

这个游戏曾经风靡一时，它的解称为哈密尔顿图，并不难求。为了清楚起见，我们做一个平面图(图 9-3(*b*))，与这个十二面体的顶点和棱所组成的图同构。显然(图 9-3(*b*))中粗的边组成的圈就是一个解。符合要求的解当然不止一种，即使固定前 5 个城市(顶点)，还有 4 种不同的解。如果不要求回到原来的出发点，解(称为哈密尔顿轨)就更多了。读者可以自己试着再找出几种不同的解。

现在我们给出一般的哈密尔顿轨或哈密尔顿圈的定义：

在 G 图中，如果有一个轨(圈)经过每个顶点恰好一次，那么这个轨(圈)称为哈密尔顿轨(圈)。

哈密尔顿轨(圈)与欧拉路(回路)不同，它迄今还没有一个象样的判别方法，只是分别地给定了一些充分条件，另外给了一些必要条件，用这些条件来判定任意给定的图是否是哈密尔顿图，只能在特定的情形下才能见效。下面给出一些判别方法：

1. 若 G 是二分图，$|X|=m$，$|Y|=n$，若 $|m-n|>1$，则二分图 G 没有哈密尔顿轨；若 G 是完全二分图 $K_{m,n}$，在 $m\neq n$ 时，无哈密尔顿圈；在 $m=n$ 时，有 $[n/2]$ 条无公共边的哈密尔顿圈；

2. 若图 G 有哈密尔顿圈，从 G 中去掉若干个顶点 $V_1,V_2,\cdots,V_K$ 及与它们相邻的边得到图 G'，那么 G' 的连通分支不超过 K 个；

3. G 是顶点数 $n>3$ 的简单图。若对任两个顶点 U 和 V，$d(U)+d(V)\geqslant n-1$，则 G 中有哈密尔顿轨；若 $d(U)+d(V)\geqslant n$，则 G 中有哈密尔顿圈；或者若每个顶点的次数 $\geqslant n/2$，则 G 有哈密尔顿圈；

4. 如果 G 是一个顶点数为 n 的简单图，V 与 V' 为某一对不相邻的顶点 $d(V)+d(V')\geqslant n$，添加一条边 (V,V') 到 G 上得到图 G'，那么当且仅当 G 有哈密尔顿圈时 G' 有哈密尔顿圈。

按上述方法，依次把 G 中次数之和至少为当前顶点数的不相邻的两个顶点间添加一条边，直至没有可能再添加边为止，最后所得的图 G' 称为 G 的闭包。显然，G 有哈密尔顿圈的充分必要条件是 G' 有哈密尔顿圈。

在目前情况下，要精确地求出图 G 中的一条哈密尔顿轨(圈)，只有采用回溯法等盲目搜索的办法，运行时间很长。若读者能在程序中适当运用上述判别条件作约束的话，可能会改善一点搜索效率。

回到题目上来。所谓货郎问题 1 指的是：

一个货郎到各村去卖货，再回到出发处。每两村之间的售货时间一定。每个村都要串到且仅到一次。为其设计一条路线，使得所用旅行售货时间最短。

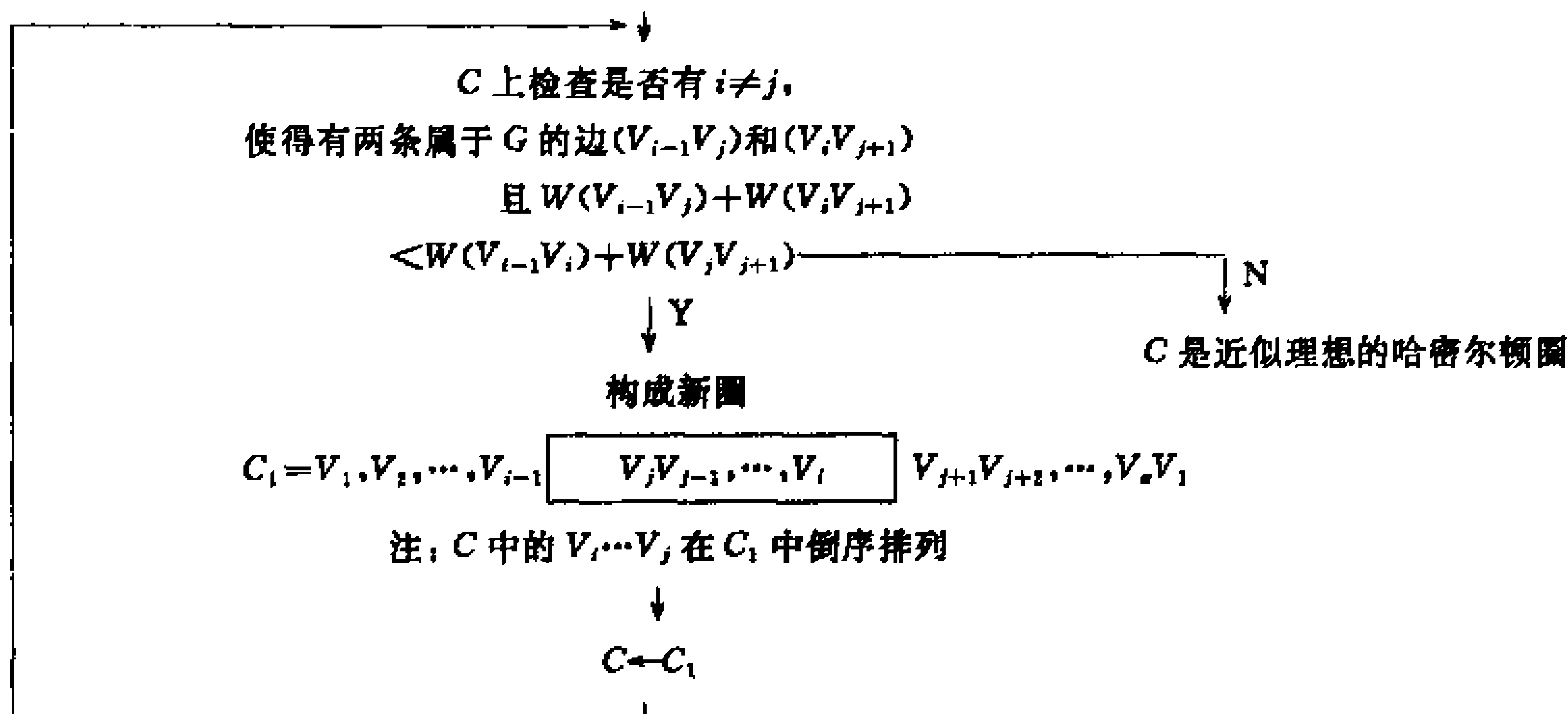
上述问题的数学模型是在加权图 G 上求一个哈密尔顿圈 C_h，使得

$$W(C_h)=\sum_{e\in C_h}W(e)=\min\{\text{各个哈密尔顿圈的权}\}$$

显然，若设 G 图每边的权为 1，则又可以成为周游世界游戏的数学模型。

上述货郎问题至今无有效算法，下面介绍一种近似算法——“改良圈算法”。

已知 $C=V_1,V_2,\cdots,V_n,V_1$ 是 G 的一个哈密尔顿圈，我们用下面的算法把它的权减小：



例如图 9-4(a)为货郎卖货的路线，各时间的售货时间作为边权在图中标出。

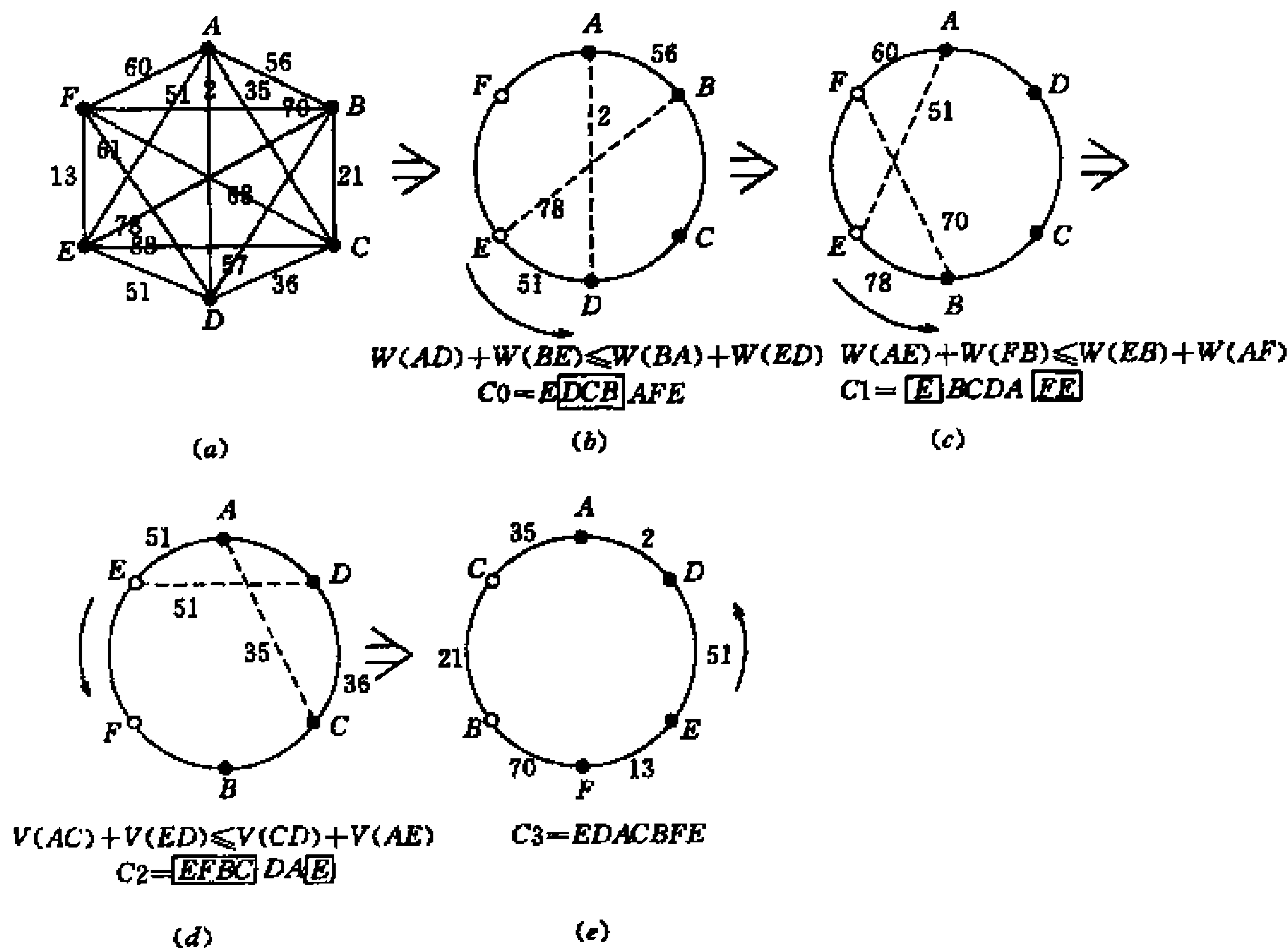


图 9-4

初始圈 $C_0=EDCBAFE$(见图 9-4(a))。图 9-4(b)，(c)，(d)，(e)给出了改良圈算法的过程。算法终止时的改良图为 $C_3=EDACBFE$，安排旅行售货的时间是

$$13+70+21+35+2+51=192(\text{小时})$$

二、程序

```
program proved_circles;
  const
    maxn          =  30;
  type
    ghtype       =  array [1..maxn,1..maxn] of integer; { 邻接矩阵类型 }
    pathtype     =  array [1..maxn] of integer;          { 改良圈类型 }
    settype      =  set of 1..maxn;
  var
    n,len        :  integer;  { 顶点数,代价和 }
    g            :  ghtype;   { 带权的邻接矩阵 }
    f            :  text;     { 文件变量 }
    v            :  pathtype; { v[i]——改良圈上第i个顶点的序号 }
  procedure read_graph;
    var
      str:string;
      i,j:integer;
    begin
      write('Graph file = ');  { 读入文件名并与文件变量连接 }
      readln(str);
      assign(f,str);
      reset(f);
      readln(f,n);              { 读入顶点数 }
      for i:=1 to n do          { 读入邻接矩阵 }
        for j:=1 to n do read(f,g[i,j]);
      close(f)
    end;
  function find_first(s:integer;m:settype):boolean;
    var
      i:integer;
    begin
      find_first:=true;
      if (m=[1..n]) and (g[v[s-1],v[1]]<>maxint)
        { 若哈密尔顿圈存在,则返回 true }
         then exit
         else for i:=1 to n do
                 if not (i in m) and (g[v[s-1],i]<>maxint)
                   { 若顶点i未扩展且圈内第s-1个顶点与顶点i有边,}
                   { 则顶点i作为圈内第s个顶点 }
                    then begin
                            v[s]:=i;
                            if find_first(s+1,m+[i]) then exit
                              { 若递归扩展下去形成哈密尔顿圈,则返回 true }
                          end;
      find_first:=false { 哈密尔顿圈不存在,返回 false }
    end;
```

```
function ft(a:integer):integer;          {      ┌-1        a=1 }
  begin                                   { ft=┤                   }
    if a=1 then ft:=n                     {      └a-1        a≠1 }
         else ft:=a-1
  end;
function nt(a:integer):integer;          {      ┌1         a=n }
  begin                                   { nt=┤                   }
    if a=n then nt:=1                     {      └a+1        a≠n }
         else nt:=a+1
  end;
procedure pcircle;
  var
    i,j,t:integer;
  begin
    for i:=1 to n do     { 若 w(i,j+1)+w(i-1,j)<w(i-1,i)+w(j,j+1), }
      for j:=1 to n do   {          则构成新的改良圈                   }
        if (i<>j) and (i<>nt(j)) and (ft(i)<>nt(j))
          then if (g[v[i],v[nt(j)]]<>maxint) and (g[v[ft(i)],v[j]]<>maxint)
          then if ((g[v[i],v[nt(j)]]+g[v[ft(i)],v[j]])
                  <(g[v[ft(i)],v[i]]+g[v[j],v[nt(j)]]))
          then begin
                 while (i<j) do
                   begin
                     t:=v[i];v[i]:=v[j];v[j]:=t;
                     inc(i);dec(j)
                   end;
                 i:=1;j:=1
               end;
    len:=0;write(v[1]:3); { 计算、打印旅行售货路线的方案和代价 }
    for i:=2 to n do
      begin
        write(v[i]:3);
        len:=len+g[v[i-1],v[i]];
      end;
    writeln(v[1]:3);
    writeln('Hamilton Length = ',len+g[v[n],v[1]])
  end;
begin
  read_graph; { 输入图 }
  v[1]:=1;
  if find_first(2,[1]) then pcircle { 若从顶点 1 出发求得哈密尔顿圈, }
                                    { 则在该圈上进行改良,并输出近似解 }
end.
```

9.3 货郎问题 2

一、再论货郎问题

货郎问题有两种定义，一种是给出了赋权无向图，要求找出经过每个顶点恰好一次而边权总和最小的哈密尔顿图。这就是 9.2 节的货郎问题 1。而货郎问题 2 放宽了每村恰好被访一次的条件，而成为不限次数地遍访每一村（即至少访问一次）而边权总和最少的回路。

显然这个问题的数学模型是在加权图 G 上求一个生成回路 C，使得

$$W(C)=\sum_{e\in C}W(e)=\min\{\text{各个生成回路的权}\}$$

这个 C 叫做理想回路。

下面给出货郎问题 2 的近似算法：

求 G 的最小生成树 T

↓

从 T 的根出发作深度优先搜索。扩展和回溯使得每条树枝都遍历了两次，得到每个顶点至少访问过 2 次的回路：

$$C_0=(V_1,V_2,\cdots,V_n,V_1);$$

↓

按 C_0 中每个顶点首次出现的次序收进哈密尔顿圈 C，终点也收进圈 C，即：

$$C=(V_{i1},V_{i2},\cdots,V_{in},V_{i1})$$

其中 $\mathrm{dfn}(V_{ij})=j$，即 C 圈上第 j 个顶点是 T 树深度搜索时，dfn 编码为 j 的顶点。

例如图 9-5(a)为一个赋权图，要求遍访每一个顶点（至少一次）而边权总和最小的哈密尔顿回路。

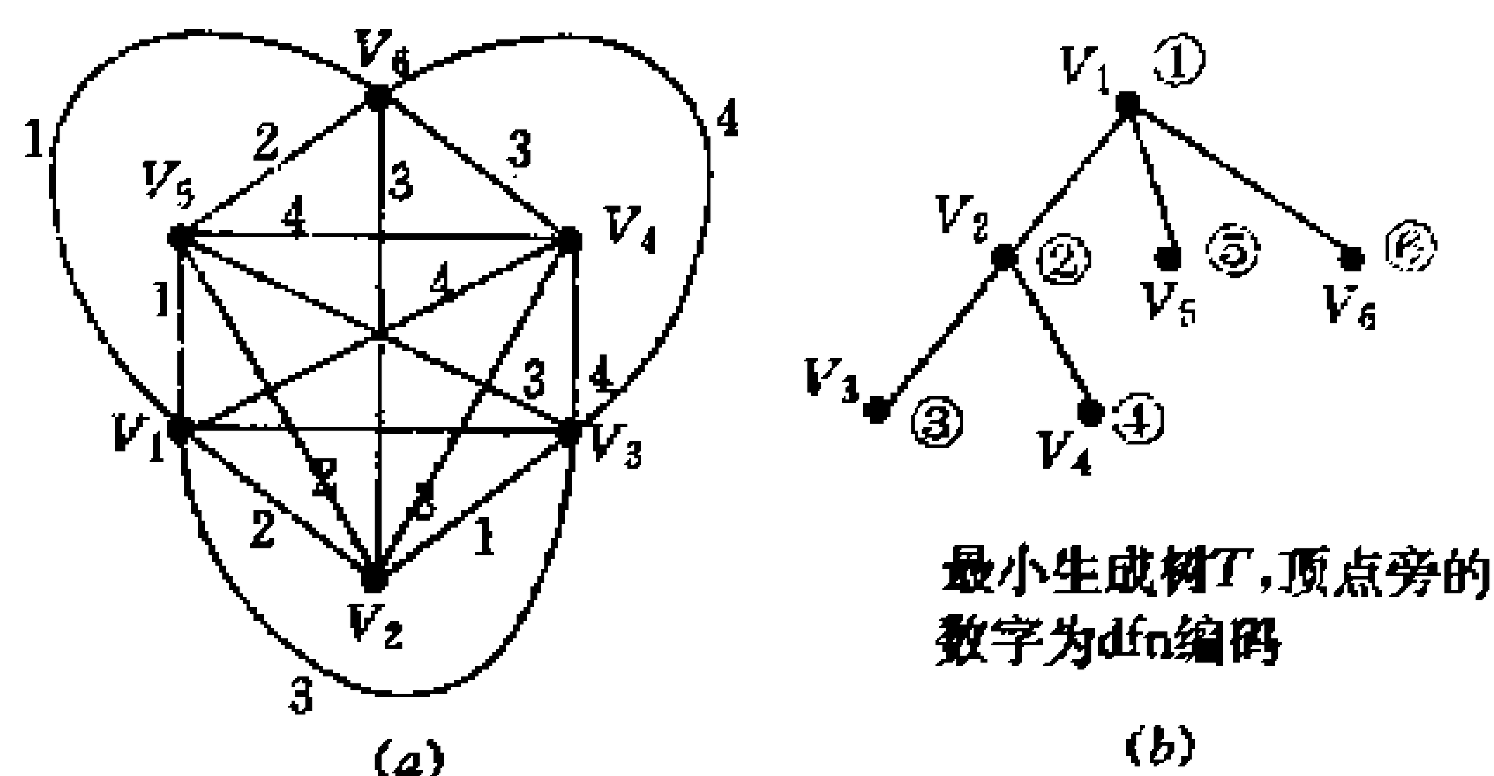


图 9-5

求图 G 的最小生成树 T（见图 9-5(b)），得出 $C_0=(V_1V_2V_3V_2V_4V_2V_1V_5V_1V_6V_1)$ 对 C_0 进行改造，得

$$C=(V_1V_2V_3V_4V_5V_6V_1)$$

C 是哈密尔顿图，总权=2+1+4+4+2+1=14。

注意，同一个近似算法，选不同的根，近似值可以不一样。例如以 V_1 为根求最小生成树 T'，然后在 T' 的基础上求近似最小权的哈密尔顿回路 C'，C' 的总权=12。读者不妨验证一下。

二、程序

```
program hamilton;
  uses
    crt;
  const
    maxn          = 50;
  type
    graphtype     = array[1..maxn,1..maxn] of integer;
    settype       = set of 1..maxn;
    ltype         = array[1..maxn] of integer;
  var
    g,path,t          : graphtype;
    { g[i,j]—顶点 i 至顶点 j 的最短路径代价                    }
    { path[i,j]—若顶点 i 到顶点 j 的最短路存在(g[i,j]>0)       }
    { 则最短路为 i→path[i,j]→j                                 }
    {                      ┌1  顶点 i 可达顶点 j               }
    { 最小生成树 t[i,j]=  ┤                                   }
    {                      └0  其它                            }
    k,father          : ltype;
    { k[i]—顶点 i 的 dfn 编码,father[i]—顶点 i 的父顶点的 dfn 编码 }
    n,cost            : integer; { n—顶点数 cost—边权和     }
    f                 : text;    { f — 文件变量             }
  procedure init;
    var i,j : integer;
        str : string;
      begin
        clrscr;
        write('file name = '); readln(str); { 读文件名 }
        assign(f,str); reset(f);  { 将文件名与文件变量连接,读准备 }
        readln(f,n);  { 读顶点数 }
        fillchar(g,sizeof(g),0);  { 邻接矩阵初始化 }
        while not eof(f) do { 读入边权 }
          begin
            readln(f,i,j,g[i,j]);
            g[j,i]:=g[i,j];
          end;
        close(f);
      end;
  procedure floyd;
  { 确定各顶点间的最短路径代价 g 和最短路径上的必经结点集合 path }
```

```
var i,j,k : integer;
  begin
    fillchar(path,sizeof(path),0);
    for k:=1 to n do
      for i:=1 to n do
        for j:=1 to n do
          if (i<>j) and (g[i,k]>0)
            and ((g[i,k]+g[k,j]<g[i,j]) or (g[i,j]=0))
          then begin
                 g[i,j]:=g[i,k]+g[k,j];
                 path[i,j]:=k;
               end;
  end;
procedure prim; { 建立最小生成树 T }
  var u          : settype;
      k,i,j,a,b,min : integer;
  begin
    u:=[1];
    fillchar(t,sizeof(t),0);
    while u<>[1..n] do
      begin
        min:=maxint;
        for i:=1 to n do
          if i in u then
            for j:=1 to n do
              if (not (j in u)) and (g[i,j]<min)
              then begin min:=g[i,j]; a:=i; b:=j; end;
        t[a,b]:=1; t[b,a]:=1;
        u:=u+[b];
      end;
  end;
procedure show(p,q:integer);
{ 打印 hamilton 回路中 p 顶点至 q 顶点的一段路 }
  begin
    if path[p,q]=0
      then begin
             writeln(p,'——',q);
             cost:=cost+g[p,q];
           end
      else begin show(p,path[p,q]); show(path[p,q],q); end;
  end;
procedure dfs;
  var v,u,p,i : integer;
  begin
    fillchar(k,sizeof(k),0); i:=0; v:=0; cost:=0;
    repeat repeat inc(v); until (v>n) or (k[v]=0);
           { 搜索一个未被访问的顶点 v }
           if v<=n then
```

```
        begin
          father[v]:=0; { 以 v 为根搜索 }
          repeat inc(i);
                 if i>1 then show(p,v);
                 { v 非根,打印 p 至 v 的一段路 }
                 p:=v; { 从 v 出发继续搜索 }
                 k[v]:=i; { 设 v 的 dfs 值 }
                 repeat u:=0;
                        repeat inc(u);
                        until (u>n) or (t[v,u]>0) and (k[u]=0);
                 { 搜索最小生成树中与 v 关联的一个未访问点 u }
                        if u<=n then
                 { 若 u 存在,则设 u 的父亲顶点为 v,继续从 u 出发搜索 }
                          begin
                            father[u]:=v;
                            v:=u;
                          end
                        else v:=father[v];
                 { 若 v 顶点是叶子,则回溯至 v 的父顶点 }
                 until (u<=n) or (v=0);
                 { 直至从 v 扩展出 u 或若以 v 为根的树全部扩展完 }
          until v=0; { 直至以 v 为根的树全部扩展 }
        end;
    until v>n; { 直至所有顶点搜索过 }
    writeln('total cost : ',cost); { 打印最佳路径代价 }
  end;

begin
  init;  { 读入图 }
  floyd; { 求各顶点间的最佳路径 }
  prim;  { 建立最小生成树 T }
  dfs;
  { 深度优先搜索 T 树,对访问序列 C0 进行改造,得出哈密尔顿回路 }
end.
```

9.4 排序问题

一、问题及其解法

今有工作 $j_1, j_2, \cdots, j_n$,要在同　　　　　　　　,从 j_i 到 j_j 的机器调整时间为 n 项工作排队,使得 $\sum t_{ij} = \min$。

上述问题至今仍无有效算法,我们这里只能给出一个近似算法。在讲算法之先引出底图和竞赛图的定义:

1. 以 G 为底图的有向图中必有长为 $X(G)-1$ 的有向轨。显然，由于 $X(K_n)=n$，所以竞赛图中有长为 $X(K_n)-1=n-1$ 的有向轨，即竞赛图中有有向哈密尔顿轨(含一切顶点的有向轨)。

2. G 是无环有向图，则底图中含有一个独立集，从 S 内的顶点出发，通过长度至多是 2 的有向轨，可以到达除 S 外的任一顶点。显然由于 K_n 的最大独立集为一个顶点构成，因此竞赛图中含有一个顶点，从该顶点出发，通过长至多为 2 的有向轨，可以到达任何顶点。

下面，我们转入近似算法的分析：

1. 以工作 $j_1,j_2,\cdots,j_n$ 为有向图 G 的顶点集，当且仅当 $t_{ij}\leqslant t_{ji}$ 时，做有向边 $<j_i,j_j>$，于是得到的有向图中含生成竞赛图；

2. 在有向图 G 中求有向哈密尔顿轨，依此哈密尔顿轨上顶点的顺序按排工作序列。例如，机器调整矩阵为

	j_1	j_2	j_3	j_4	j_5	j_6
j_1	0	5	3	4	2	1
j_2	1	0	1	2	3	2
j_3	2	5	0	1	2	3
j_4	1	4	4	0	1	2
j_5	1	3	4	5	0	5
j_6	4	4	2	3	1	0

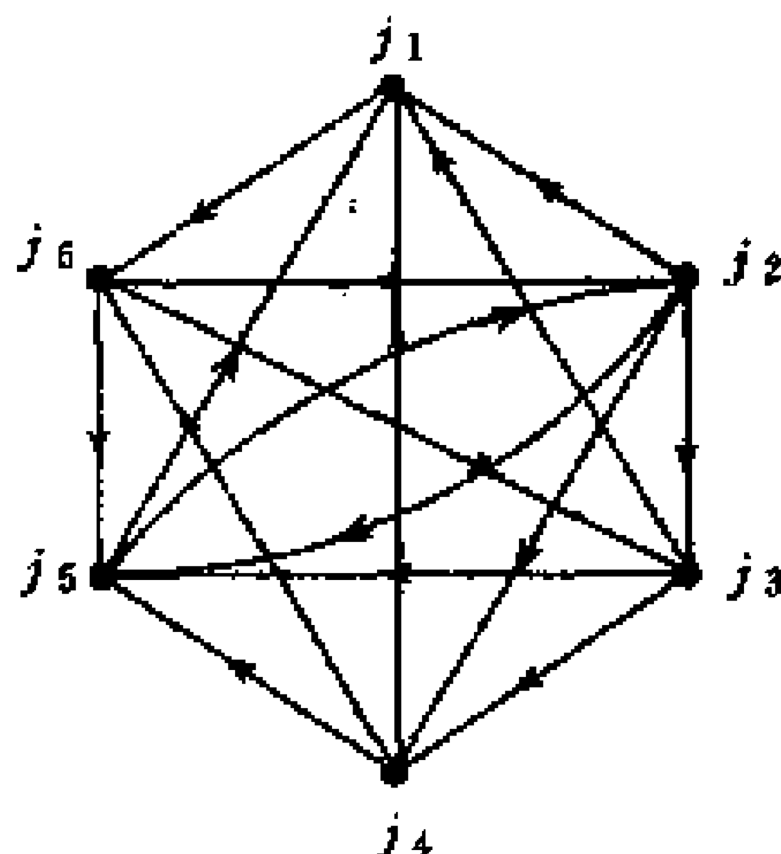


图 9-6

求这 6 项工作如何安排工序，使总加工时间最少。

我们构作有向图 G，以 $j_1,j_2,\cdots,j_6$ 为顶点集，当且仅当 $t_{ij}\leqslant t_{ji}$ 时，连一条有向边 $<j_i,j_j>$(见图 9-6)。

从图 G 中求一条有向哈密尔顿轨为 $<j_1,j_6,j_3,j_4,j_5,j_2>$，所需调机时间为

$$1+2+1+1+3=8$$

若用自然顺序 $j_1j_2j_3j_4j_5j_6$，则所需调机时间为

$$5+1+1+1+5=13$$

二、最佳排序问题的程序

```
program gong_zuo;
```

```
uses
  crt;

const
  maxn              = 50;

type
  graphtype         = array[1..maxn,1..maxn] of integer; { 邻接矩阵类型 }
  ltype             = array[1..maxn] of integer;           { 圈类型 }
  settype           = set of 1..maxn;

var
  g                 : graphtype; { 带权的邻接矩阵 }
  l                 : ltype;      { l[i]——圈内第i个顶点的序号 }
  n                 : integer;    { 顶点个数 }
  f                 : text;       { 文件变量 }

procedure init;
  var i,j : integer;
      str : string;
    begin
      clrscr; write('filename='); readln(str);
      { 输入文件名并与文件变量连接 }
      assign(f,str); reset(f); readln(f,n);
      { 读准备,读入顶点数和邻接矩阵 }
      for i:=1 to n do
        for j:=1 to n do
          read(f,g[i,j]);
      close(f);
    end;

function hamilton(lev:integer;s:settype):boolean;
  var
    i,j : integer;
  begin
    hamilton:=true;
    if lev=n+1 then exit; { 若哈密尔顿圈形成,则返回 true }
    j:=l[lev-1];               { 恢复前一个顶点的序号 }
    for i:=1 to n do
    { 以顶点j为一端,按ₜ≤ᵢⱼ的要求寻找有向边<j,i> }
      if not (i in s) and (i<>j) and (g[j,i]<=g[i,j])
        then begin
               l[lev]:=i; { 顶点i进入哈密尔顿圈 }
               if hamilton(lev+1,s+[i]) then exit;
               { 若递归搜索圈的下一个顶点成功,则返回 true }
             end;
    hamilton:=false; { 哈密尔顿圈不存在,返回 false }
  end;

procedure main;
  var i,j,cost : integer;
    begin
      for i:=1 to n do
```

```
        begin
          l[1]:=i; { 从 vi 开始搜索哈密尔顿圈 }
          if hamilton(2,[i])
          { 若搜索成功,则计算调机时间,打印退出 }
            then begin
                    write(i); cost:=0;
                    for j:=2 to n do
                      begin write(l[j]:3); inc(cost,g[l[j-1],l[j]]); end;
                    writeln;
                    writeln('cost : ',cost);
                    exit;
                 end;
        end;
    end;
begin
  init; { 输入图 }
  main; { 计算和输出最佳工作排序 }
end.
```

青少年计算机竞赛指导丛书

